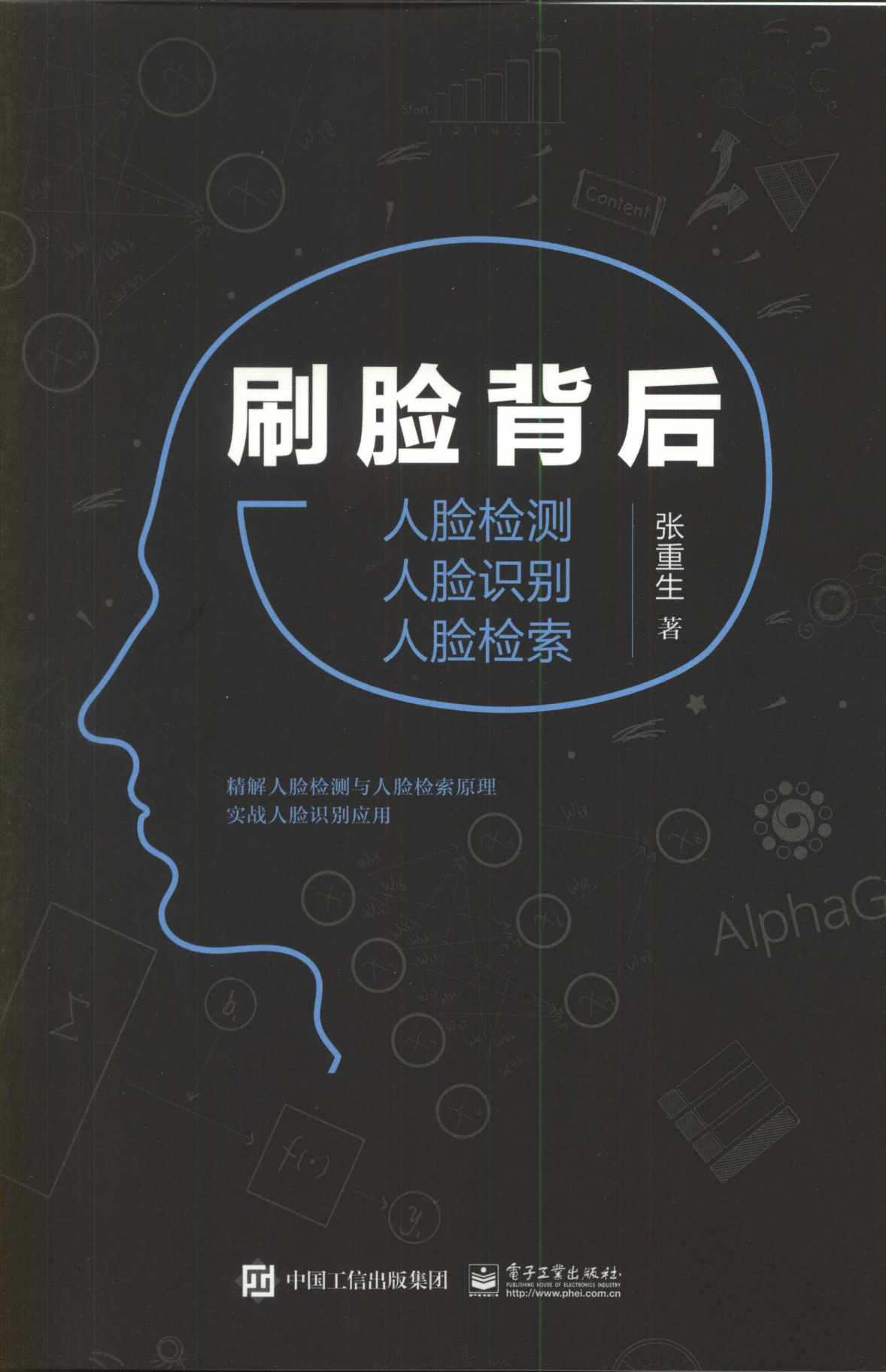
版权注意事项：
1、书籍版权归作者和出版社所有
2、本PDF仅限用于个人获取知识，进行私底下的知识交流
3、PDF获得者不得在互联网上以任何目的进行传播
4、如觉得书籍内容很赞，请购买正版实体书，支持作者
5、请于下载PDF后24小时内删除本PDF。



# 刷脸背后

## 人脸检测
## 人脸识别
## 人脸检索

张重生 著

精解人脸检测与人脸检索原理
实战人脸识别应用

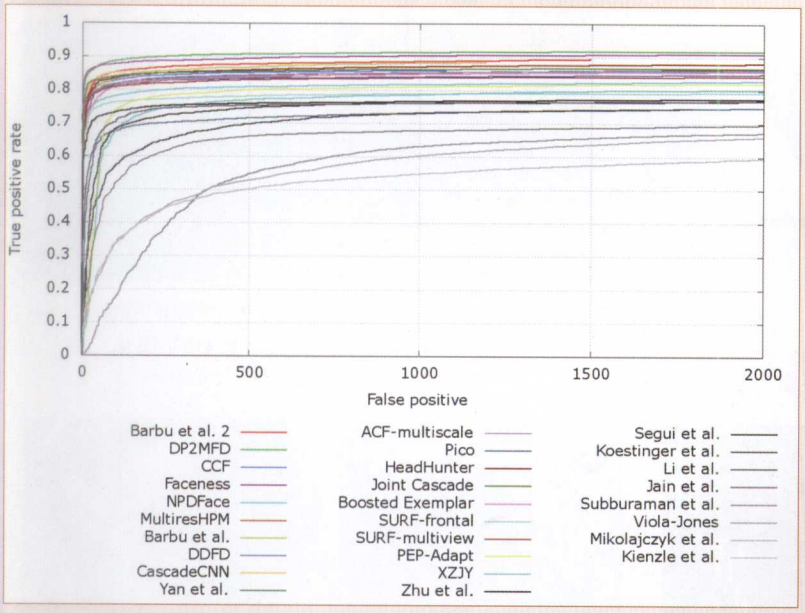
中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn



**张重生，**1982年9月生，博士，教授，硕士生导师，河南大学大数据团队带头人。研究领域为大数据分析、深度学习、数据挖掘、数据库、实时数据分析。

博士毕业于 INRIA,France（法国国家信息与自动化研究所），获得优秀博士论文荣誉。2010年8月至2011年3月，在美国加州大学洛杉矶分校计算机系，师从国际著名学者Carlo Zaniolo教授进行流数据挖掘方面的研究。

十多年来，一直从事与大数据相关的研究，发表SCI/EI论文20余篇，含Information Sciences、Expert Systems with Applications、Neurocomputing、IEEE ICDM、PAKDD、SSDBM等著名期刊和国际会议论文；先后指导硕士研究生16名，作为第一发明人获得3项国家发明专利授权。出版《深度学习：原理与应用实践》、《刷脸背后：人脸检测、人脸识别、人脸检索》、《大数据分析：数据挖掘必备算法示例详解》等学术专著，以及译著《数字视频处理与分析》，作为副主编合著《大数据》等教材。




❖ 图 1-4 不同人脸检测算法在 FDDB 数据集上检测准确度的 ROC 曲线



（a）原图像

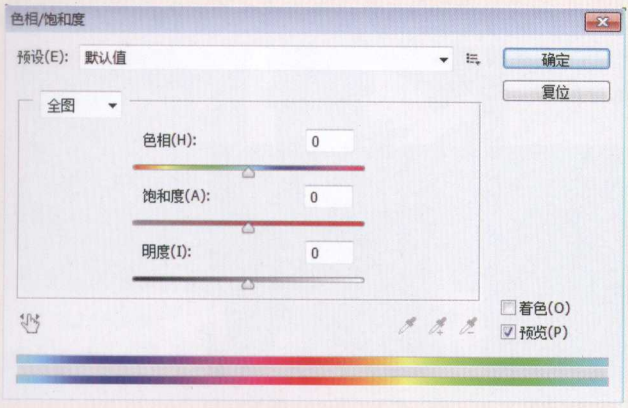


（b）饱和度为100



（c）饱和度为-100

❖ 图 2-4 同一图片不同饱和度值的变化




❖ 图 2-5 “色相 / 饱和度”对话框



（a）原灰度图像



（b）修改对比度后的图像

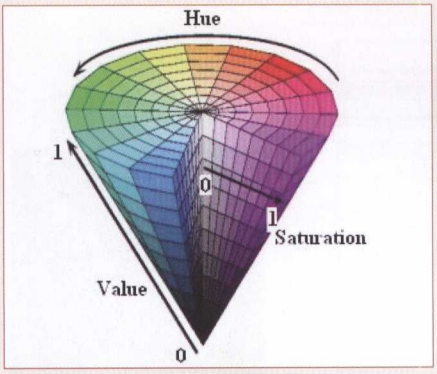


（c）原RGB图像



（d）修改对比度后的RGB图像

❖ 图 2-6 Code2 的运行结果




❖ 图 2-10 HSV 颜色空间模型



（a）原RGB图像



（b）转换后的HSV图像

✧ Figure1：原 RGB 图像和转换后的 HSV 图像



H分量



S分量



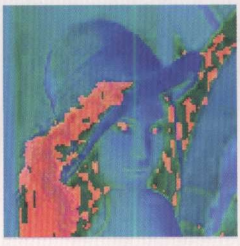
V分量

✧ Figure2：转换后的 HSV 图像的 H、S、V 分量图

❖ 图 2-11 Code4 的运行结果



（a）原RGB图像



（b）转换后的HSV图像



（c）转换后的RGB图像

❖ 图 2-12 Code5 的运行结果



（a）原真彩色图像



（b）转换后灰度图像1



（c）转换后灰度图像2

❖ 图 2-19 Code12 的运行结果




（a）原索引图像　　（b）转换后的灰度图像

❖ 图 2-20　Code13 的运行结果




（a）原灰度图像　　（b）转换后的二值图像

✧ Figure1：原灰度图像和转换后的二值图像的比较




（a）原RGB图像　　（b）转换后的二值图像

✧ Figure2：原 RGB 图像和转换后的二值图像的比较



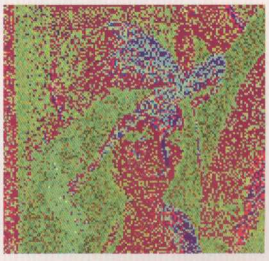

（a）原索引图像　　（b）转换后的二值图像

✧ Figure3：原索引图像和转换后的二值图像的比较

❖ 图 2-25　Code18 的运行结果



（a）原RGB图像



（b）转换后的索引图像1

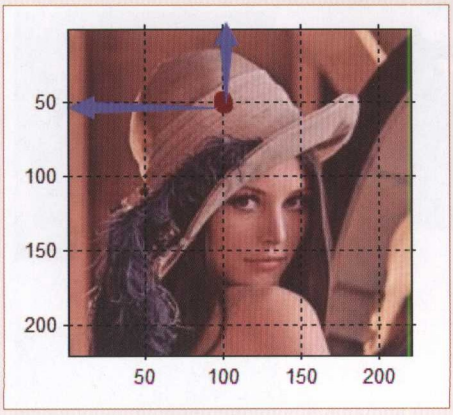


（c）转换后的索引图像2

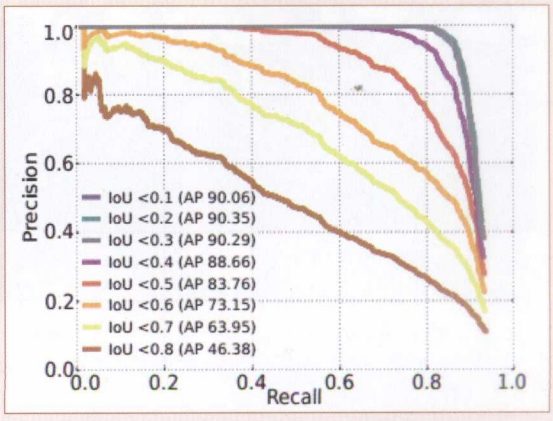


（d）转换后的索引图像3

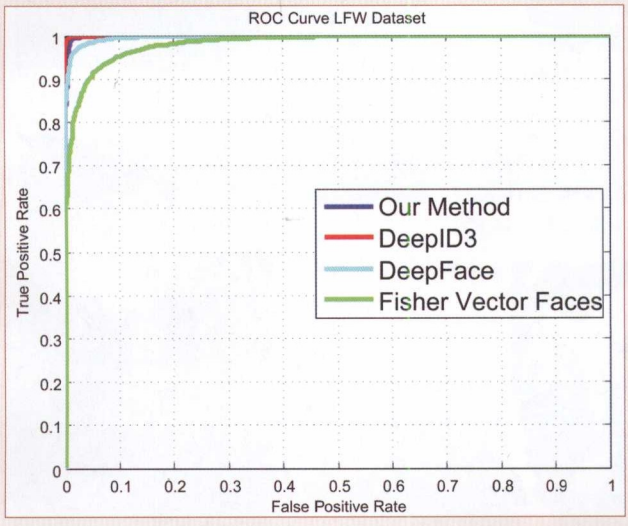
❖ 图 2-23　Code16 的运行结果



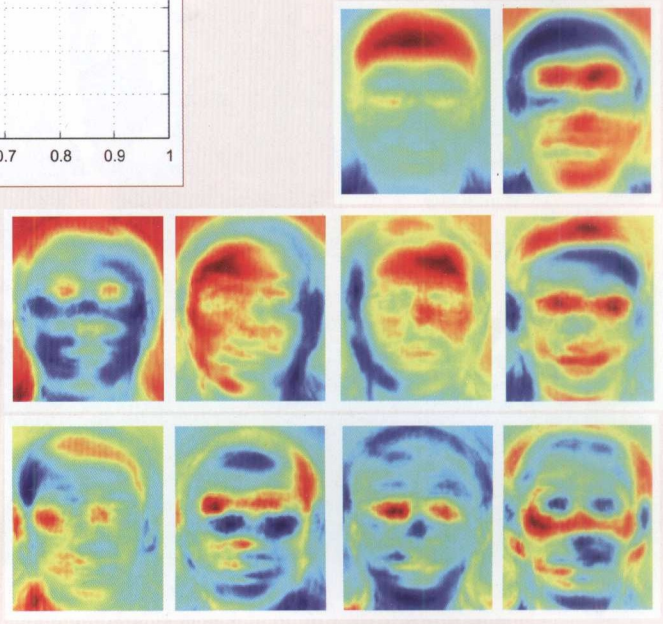

❖ 图 2-26 图像在 MATLAB 中的坐标形式




❖ 图 3-3 DPM 在 Pascal Faces 数据集上使用不同重叠阈值的检测结果




❖ 图 6-7 在 LFW 数据集上测试的结果



❖ 图 6-8 使用了 JET 模式色图的 10 个 Eigenfaces

# 刷脸背后

## 人脸检测 人脸识别 人脸检索

张重生 著

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

## 内 容 简 介

人脸识别是当今热门的研发方向，在安防、金融、旅游等领域具有十分广泛的应用。本书全面、系统地介绍“刷脸”背后的技术，包括人脸检测、人脸识别、人脸检索相关的算法原理和实现技术。本书中讲解的算法具有很强的可操作性和实用性。通过学习本书，研究人员、工程师能够在 3～5 个月内，系统了解、掌握人脸检测、人脸识别、人脸检索相关的原理和技术。

本书内容新颖、层次清晰，适合高校教师、研究人员、研究生、高年级本科生、人脸识别爱好者使用。

**未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。**
**版权所有，侵权必究。**

**图书在版编目（CIP）数据**

刷脸背后：人脸检测　人脸识别　人脸检索 / 张重生著. —北京：电子工业出版社，2017.8

ISBN 978-7-121-32138-2

Ⅰ. ①刷… Ⅱ. ①张… Ⅲ. ①面－图象识别－研究 Ⅳ. ①TP391.41

中国版本图书馆 CIP 数据核字（2017）第 161123 号

策划编辑：董亚峰
责任编辑：董亚峰
特约编辑：赵树刚　赵海军等
印　　刷：北京季蜂印刷有限公司
装　　订：北京季蜂印刷有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：720×1000　1/16　印张：15.25　字数：348 千字　彩插：2
版　　次：2017 年 8 月第 1 版
印　　次：2017 年 12 月第 2 次印刷
定　　价：48.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：QQ3502629。

# 前　言

我们正处于“刷脸”的时代，越来越多的“刷脸”应用开始出现。例如，北京西站的刷脸检票、厦门景点的刷脸验票、余额宝的刷脸认证等。初学者如果想进行人脸识别相关的研究和开发，那么他们应该阅读什么书籍呢？

“刷脸”背后的技术，不仅仅是人脸识别，亦需要人脸检测和人脸检索等技术提供支撑。目前，市场上有少部分人脸识别的书籍，而专门讲解人脸检测和人脸检索技术的书籍则更少。近年来，笔者及其团队在从事人脸检测、人脸识别、人脸检索相关的研究时，查阅了很多国内外的参考资料，到目前为止，尚未见到一本能够全面涵盖“刷脸”应用所涉及的人脸检测、人脸识别和人脸检索相关技术且具有实战参考价值的书籍。其中的一个主要原因可能是刷脸技术的商业价值高。

本书按照“刷脸”应用开发时所需技术的先后顺序，通过原理、案例、实战的方式，分别讲解了“刷脸”应用需要掌握的三大技术：人脸检测、人脸识别和人脸检索。更为重要的是，本书高度注重实战应用，每一个算法都通过具体程序讲解算法的使用、实验设计，以及实验结果。读者不但能够了解每个算法的原理，而且能够掌握应用开发的实战技能。

本书的目标是作为通用、普及性强、可操作性强的人脸识别的书籍，方便研究人员、工程师、研究生、计算机专业的高年级本科生，快速上手并全面、深入理解，扎实掌握“刷脸”应用相关的理论和算法，帮助读者快速入门，理解“刷脸”应用背后的核心技术与算法，并切实掌握“刷脸”应用开发所需的实战技术。

本书主编为张重生，副主编为王弯弯、王朋友、赵冬冬。于珂珂、彭国雯、裴宸平等研究生对本书的编写、实验部分的验证提供了一定的帮助，在此致谢。

笔者自知才疏学浅，仅略知人脸检测、人脸识别、人脸检索之皮毛。书中错谬之处在所难免，如蒙读者不吝告知（邮箱：chongsheng.zhang@yahoo.com，微信号：A13938613173），将不胜感激。

张重生

2017 年 4 月

# 目　录


第 1 章　人脸检测、人脸识别与人脸检索概述 ..... 1
1.1　人脸检测、人脸识别与人脸检索的应用场景 ..... 2
1.1.1　当前应用 ..... 3
1.1.2　未来应用 ..... 5
1.2　人脸检测、人脸识别与人脸检索常用的数据集 ..... 5
1.2.1　LFW 数据集 ..... 5
1.2.2　FDDB 数据集 ..... 6
1.2.3　Wanwan1 数据集 ..... 7
1.2.4　Wanwan2 数据集 ..... 8
1.3　OpenCV 的简介、安装与使用 ..... 8
参考文献 ..... 15
第 2 章　图像处理基础 ..... 16
2.1　数字图像处理的基本概念 ..... 16
2.1.1　像素 ..... 17
2.1.2　分辨率 ..... 17
2.1.3　图像的色调、亮度和饱和度 ..... 19
2.1.4　图像的对比度 ..... 22
2.1.5　图像的纹理 ..... 23
2.2　颜色空间 ..... 26
2.2.1　RGB 颜色空间 ..... 26
2.2.2　HSV 颜色空间 ..... 27
2.2.3　YUV 颜色空间 ..... 27
2.2.4　颜色空间的转换 ..... 28
2.3　数字图像处理的基本操作 ..... 32
2.3.1　图像的读取 ..... 32
2.3.2　图像的显示 ..... 34

2.3.3 图像的修改 ..... 35
2.3.4 图像的保存 ..... 36
2.3.5 获取图像的基本信息 ..... 37
2.4 图像类型及转换 ..... 38
2.4.1 图像类型 ..... 39
2.4.2 图像类型的转换 ..... 39
2.5 图像变换处理 ..... 48
2.5.1 图像的平移 ..... 48
2.5.2 图像的旋转 ..... 51
2.5.3 图像的缩放 ..... 52
2.5.4 图像的剪切 ..... 55
2.5.5 图像的翻转 ..... 58
2.6 图像的噪声和滤波 ..... 60
2.6.1 常见的噪声模型 ..... 60
2.6.2 经典的去噪算法 ..... 64
第 3 章 人脸检测实战 ..... 67
3.1 DPM 人脸检测算法 ..... 67
3.1.1 DPM 人脸检测算法的使用 ..... 68
3.1.2 DPM 人脸检测算法的原理 ..... 69
3.1.3 DPM 人脸检测算法的检测结果 ..... 73
3.2 LAEO 人脸检测算法 ..... 74
3.2.1 LAEO 人脸检测算法的使用 ..... 74
3.2.2 LAEO 人脸检测算法的原理 ..... 75
3.2.3 LAEO 人脸检测算法的检测结果 ..... 77
3.3 Viola&Jones 人脸检测算法 ..... 79
3.3.1 Viola&Jones 人脸检测算法的使用 ..... 79
3.3.2 Viola&Jones 人脸检测算法的原理 ..... 79
3.3.3 Viola&Jones 人脸检测算法的检测结果 ..... 82
参考文献 ..... 83
第 4 章 基于深度学习的人脸检测算法 ..... 84
4.1 CNN Facial Point Detection 人脸检测算法 ..... 84
4.1.1 CNN Facial Point Detection 人脸检测算法的使用 ..... 85

4.1.2 CNN Facial Point Detection 人脸检测算法的原理 ..... 85
4.1.3 CNN Facial Point Detection 人脸检测算法的检测结果 ..... 86
4.2 DDFD 人脸检测算法 ..... 87
4.2.1 DDFD 人脸检测算法的使用 ..... 87
4.2.2 DDFD 人脸检测算法的原理 ..... 88
4.2.3 DDFD 人脸检测算法的检测结果 ..... 89
4.3 人脸检测算法融合 ..... 90
参考文献 ..... 92
第 5 章 基于 Fast R-CNN 的人脸检测 ..... 94
5.1 Fast R-CNN 简介 ..... 94
5.2 Fast R-CNN 的特点和结构 ..... 95
5.3 Fast R-CNN 的使用 ..... 96
5.4 数据集的预处理 ..... 97
5.5 EdgeBoxes 的使用 ..... 98
5.6 使用 EdgeBoxes 提取 object proposal ..... 99
5.7 基于 Fast R-CNN 训练人脸检测网络模型和测试 ..... 100
5.7.1 训练阶段 ..... 100
5.7.2 测试阶段 ..... 106
5.7.3 评估阶段 ..... 108
5.7.4 优化阶段 ..... 111
参考文献 ..... 112
第 6 章 人脸识别实战 ..... 113
6.1 DeepID 算法 ..... 114
6.1.1 DeepID 算法的原理 ..... 114
6.1.2 DeepID 算法的流程 ..... 116
6.1.3 DeepID 算法的结果 ..... 126
6.2 VGG Face Descriptor 算法 ..... 128
6.2.1 VGG Face Descriptor 算法的原理 ..... 128
6.2.2 VGG Face Descriptor 算法的实现 ..... 129
6.2.3 VGG Face Descriptor 算法的结果 ..... 131
6.3 OpenCV 中的 3 种人脸识别算法 ..... 132
6.3.1 Eigenfaces ..... 132

6.3.2 Fisherfaces ..... 140
6.3.3 Local Binary Patterns Histograms ..... 148
6.4 人脸识别算法对比分析 ..... 152
6.5 小结 ..... 153
参考文献 ..... 155
第 7 章 人脸检索实践 ..... 157
7.1 人脸检索简介 ..... 157
7.2 计算人脸相似度的方法 ..... 158
7.2.1 欧氏距离 ..... 159
7.2.2 余弦相似度 ..... 159
7.3 查询处理算法 ..... 161
7.4 评价人脸检索结果的标准 ..... 161
7.5 PHash 算法 ..... 161
7.5.1 PHash 算法的使用 ..... 162
7.5.2 PHash 算法原理 ..... 162
7.5.3 PHash 算法实现 ..... 162
7.5.4 PHash 算法的实验数据、实验结果及分析 ..... 164
7.6 DHash 算法 ..... 168
7.6.1 DHash 算法的使用 ..... 168
7.6.2 DHash 算法原理 ..... 168
7.6.3 DHash 算法实现 ..... 169
7.6.4 DHash 算法的实验数据、实验结果及分析 ..... 170
7.7 PCA 算法 ..... 173
7.7.1 PCA 算法的使用 ..... 173
7.7.2 PCA 算法原理 ..... 174
7.7.3 PCA 算法实现 ..... 175
7.7.4 PCA 算法的实验数据、实验结果及分析 ..... 177
7.8 BoF 特征 ..... 181
7.8.1 BoF-SIFT 算法的使用 ..... 182
7.8.2 BoF-SIFT 算法原理 ..... 182
7.8.3 BoF-SIFT 算法实现 ..... 182
7.8.4 BoF-SIFT 算法的实验数据、实验结果及分析 ..... 188

7.9 用于图像快速检索的 KD-Tree 索引 ..... 190
7.9.1 FLANN 算法的使用 ..... 191
7.9.2 KD-Tree 的创建与查询处理 ..... 191
7.9.3 FLANN 中 KD-Tree 的算法实现 ..... 192
7.9.4 FLANN 算法的实验数据、实验结果及分析 ..... 194
7.10 Gabor 算法 ..... 195
7.10.1 Gabor 算法的使用 ..... 196
7.10.2 Gabor 算法原理 ..... 196
7.10.3 Gabor 算法实现 ..... 199
7.10.4 Gabor 算法的实验数据、实验结果及分析 ..... 204
7.11 HOG 算法 ..... 208
7.11.1 HOG 算法的使用 ..... 209
7.11.2 HOG 算法原理 ..... 209
7.11.3 HOG 算法实现 ..... 210
7.11.4 HOG 算法的实验数据、实验结果及分析 ..... 212
7.12 深度学习特征 ..... 215
7.12.1 深度学习算法的使用 ..... 215
7.12.2 深度学习算法原理 ..... 215
7.12.3 深度学习算法实现 ..... 216
7.12.4 深度学习算法的实验数据、实验结果及分析 ..... 216
参考文献 ..... 220
**第 8 章 人脸检测商业软件及其应用示例 ..... 222**
8.1 人脸检测商业软件之 VeriLook ..... 222
8.2 人脸检测商业软件之 Face++ ..... 226
8.3 各种人脸检测算法的对比分析 ..... 229
8.4 视频中的人脸检测与追踪 ..... 231
参考文献 ..... 234

# 第1章

# 人脸检测、人脸识别与人脸检索概述

视频和图像中人脸的检测、识别和检索，这些图像处理技术和应用在当今的科技发展中发挥着越来越重要的作用。例如，在公安、交通应用中，人群密度高的区域、城市的道路上都已部署了大量的监控摄像头（分辨率多为200万像素的高清数字摄像机），实现对目标人物（如可疑人物、犯罪嫌疑人）及其活动轨迹的全自动识别和检索，在公安和交通等应用中具有极其强烈的现实需求（目前主要通过人眼对视频进行人工分析，即办案人员用眼睛盯着视频、逐帧查看、反复比对，这一过程通常极为耗时）。要想实现对这些视频的全自动、智能分析，就需要准确地对人脸进行检测、识别和检索。而人脸检测、人脸识别和人脸检索，是当今智能识别的商业应用中迫切需要解决的现实问题。本书以实例为驱动，从应用的角度全面介绍、讲解、分析人脸检测、人脸识别和人脸检索问题中的经典算法及其应用。

“人脸检测”和“人脸识别”是两个容易混淆的概念。通俗地讲，人脸检测是根据肤色等特征定位人脸区域；人脸识别是识别这个人到底是谁。换句话说，需要首先在图像中检测出人脸的区域，即人脸检测，然后再使用人脸识别的算法，

识别出图像中的人到底是谁。而人脸检索，指给定一个或多个包含人脸的输入图像，从图像库或视频库中检索包含所输入图像中的人脸的那些图像。人脸检测和人脸检索通常都是非监督学习的过程，即图像一般都没有标签/分类；而人脸识别是有监督学习的过程，需要使用一定数量的有标签的图像训练分类模型。近年来，随着深度学习的发展，使用大量有标记的人脸图像训练的人脸识别模型，能够更加准确地提取人脸特征，用于人脸的检索、检测中。

需要说明的是，尽管现有算法在 LFW 等公开数据集上取得了 96%甚至更高的准确度，但人脸检测、人脸识别与人脸检索仍然是一个尚未很好解决的实际问题。事实上，尽管 LFW 数据集比原有的公开数据集更加有挑战性、更加接近真实应用，但其仍然只是一个非常简单和初级的数据集，远不能代表真实应用场景。由于表情、年龄变化、光线变化、不同姿态、不同图像分辨率等因素，现有的技术还达不到完全好用的程度。事实上，多数相关产品都要求正脸照，很多商业软件还需要手动采集不同光照、有无眼镜时人的图像。

## 1.1 人脸检测、人脸识别与人脸检索的应用场景

人脸检测是计算机视觉中的一个重要方向，也是一个和人们生活息息相关的研究方向，因为人脸是人最重要的外貌特征。利用计算机智能地检测、识别并分析人脸的特征信息，并利用这些信息进行进一步处理，可以使人们的生活更加方便化、安全化和自动化。

人脸检测技术在最近几年取得了突破性的进步。最初，人脸检测只能在无背景图片中检测出人脸的位置。现在，部分人脸检测技术可以准确地检测出自然场景下多种角度的人脸；可以判断两张人脸是否为同一个人；可以根据人脸分析人的年龄、性别和表情等。在人脸检测技术发展的过程中，出现了一系列的应用。本节就对基于人脸检测、人脸识别和人脸检索技术的应用做一个简单的概述。

### 1.1.1　当前应用

#### 1. 人脸检测应用之面部分析

微软公司在 2015 年推出了一个有趣的网站，即 how-old.net，该网站根据用户提供的照片分析出人的表面年龄和性别。该项目是微软工程师为了测试面部分析技术而发布的，并不是正式的项目，但自发布的短短几个小时内，使用的用户就达到了 3.5 万人。其检测结果和实际的年龄、性别可能会不一致，但整体上来说，比较接近。在国内，三个“85 后”学生创建了 Face++，它能够提供准确的面部分析技术，不仅可以分析出人脸的年龄、性别，还可以分析出种族和表情信息，其中性别准确率达到了 96%。Face++为许多应用提供服务，比如，将 Face++的人脸检测和识别技术嵌入在美颜相机和美图秀秀，从而更加准确地定位人脸中需要美化的区域，达到自动美颜的效果。现在一些手机自带的摄像功能可以在拍摄时自动识别年龄和性别，如小米 4。

#### 2. 人脸识别应用之刷脸考勤

出勤记录对公司员工来说是必不可少的一件事情，传统的考勤是打卡签到或者点名签到。自从人脸检测和识别技术出现以来，逐渐出现了刷脸考勤机，该机器可以快速准确地识别员工、记录出勤信息。比如，某公司的刷脸考勤系统，首先将公司内部员工的人脸信息录入该系统，然后将嵌入人脸检测技术的摄像头连接到该系统，之后便可进行刷脸考勤。员工面对该考勤机时，其摄像头会自动捕捉到员工的人脸区域，并将获得的人脸区域和员工信息数据库进行比对，识别出该员工的姓名，并且记录该员工到达的时间，这便代替了常规的打卡签到。现在一些教室也使用刷脸考勤代替点名考勤，这在很大程度上提高了考勤率，降低了代签到的现象。

#### 3. 人脸识别应用之公安破案

现在一些城市的车站、海关、机场等地安置的摄像头都嵌入了人脸检测技术，公安会提前将一些犯罪分子的脸部信息录入系统，对视频中出现的每个人脸进行捕捉、分析，判断是否为犯罪分子。如果是犯罪分子，则系统能够自动发出警报，并追踪他们，从而加快公安破案。在 2015 年 6 月份，一名潜逃至中国台湾 11 年的福建籍女子晏某回中国大陆办理逾期的通行证，入境时，被海关的人脸识别技术捕捉到，发现该女子和晏某极其相似，经过公安的进一步侦查，发现其就是当

年的晏某，最后将晏某抓捕归案。许多城市的公安系统已经使用该技术破获了许多案件，保障了城市的安全。

### 4．人脸检测应用之刷脸验票

2016 年 1 月，厦门的邮轮中心开始使用刷脸验票机，乘客将二代身份证放在指定区域后，识别机上会显示二代身份证上的照片，并通过摄像头捕捉游客的人脸，比对当前游客的人脸与二代身份证中的人脸，等到机器识别后，会播报“验证成功，请通过”；否则会播报“人员与证件不符”。该机器识别快，仅需要 5 秒钟即可，并且正确率高达 99%，杜绝了冒用他人身份证件的事情发生。

### 5．人脸检测应用之刷脸登录

阿里巴巴旗下的支付宝已经可以刷脸登录。在支付宝中，点击“安全设置”→“刷脸”，开启刷脸登录，然后根据提示按要求对脸部进行拍摄，录入成功后，退出账号便可使用刷脸进行登录。其大致流程如图 1-1 所示。在网络时代，每个人都在许多网站、APP 上有自己的账号和密码，如微博的账号和密码、支付宝的账号和密码、微信的账号和密码、美团外卖的账号和密码等。为了安全起见，这些密码又各不相同，对于人类有限的记忆来说，很容易忘记这些密码。刷脸登录便可解决这些问题，登录时，这些软件自动对脸部进行检测，确认是本人后，便提示登录成功，否则提示登录失败。




图 1-1　刷脸登录的设置流程

### 1.1.2　未来应用

虽然已经出现了一些刷脸登录、刷脸验票的机器，但是这些机器还没有广泛普及到普通大众中去。随着人脸检测和识别准确率的提高和人脸数据库的增大，以及从人脸中获取更多的信息（不再局限于性别、年龄和表情，也能获得姓名、职业等基本信息），未来刷脸登录会出现在各个 APP 中，如微信、微博、QQ、淘宝等。刷脸验票也会应用到火车站、机场、汽车站、旅游景区等多个场所。刷脸支付则会出现在银行、电商、支付宝、ATM 机等金融渠道中。人脸检测还可以应用于广告中，在露天的广告牌前安置嵌入人脸检测技术的软件，该软件根据行人在广告牌前的表情、浏览时间等，分析行人是否对该广告感兴趣以及是否适合该广告。如果判定该行人需要这个广告，则向该行人的邮箱投递该广告；否则不向该行人投递广告。

未来，人脸检测应用会逐渐深入到交通、网络、安全、经济等领域，广泛应用在各行各业中，会成为人们生活息息相关的一部分。

## 1.2　人脸检测、人脸识别与人脸检索常用的数据集

人脸检测和人脸识别算法都需要在大量的人脸数据集上验证、测试，当前比较经典和常用的人脸数据集有 LFW 和 FDDB。本书还收集了另外两个数据集，分别命名为 Wanwan1 和 Wanwan2。本节分别对这四个数据集进行介绍。

### 1.2.1　LFW 数据集

LFW（Labeled Faces in the Wild）数据集[1]是由美国马萨诸塞大学阿姆特分校收集整理并维护的，是为了在不受限制的环境下研究人脸识别问题而设计的[2]。该数据集中有 13 233 张图片，这些图片均从网上收集而来，得到这些原始图像后，经过处理，最后数据集中每张图片的尺寸都是 250×250 像素的，并且每张图片都以图像中人物的名字命名，不同人的图片被放在不同的文件夹中。一共有 5749 个文件夹，即 5749 个人。其中，4069 个人只有一张图片，而 1680 个人有多张图片。如 lfw\Aaron_Peirsol 中有四张图片，命名形式如图 1-2 所示。

LFW 数据集可用于人脸检测、人脸识别和人脸检索。




图 1-2　LFW 中 Aaron_Peirsol 文件夹下的图片

## 1.2.2　FDDB 数据集

FDDB（Face Detection Data Set and Benchmark）数据集[3]由 Vidit 等人[4]收集整理，现由美国马萨诸塞大学阿姆特分校维护。设计该数据集的目的是为人脸检测算法提供一个数据集，并且在该平台上公平地比较和评估不同人脸检测算法的效果。该数据集中共有 28 736 张人脸图片，其中包含 5171 张个人脸的 2845 张图片来自数据集 Faces in the Wild[5]。此外，该数据集还对图片中的人脸区域提供了准确的椭圆形的注解框。FDDB 数据集中包含的人脸图片有不同姿势的、遮挡的、低分辨率的，甚至有失焦的，有彩色图像，也有灰度图像。如图 1-3 所示，呈现了一些 FDDB 数据集中的图片。FDDB 数据集是世界上权威的人脸检测评估平台之一，它公布了大量算法评估结果的 ROC 曲线，如图 1-4 所示。

FDDB 数据集主要用于人脸检测和人脸检索。

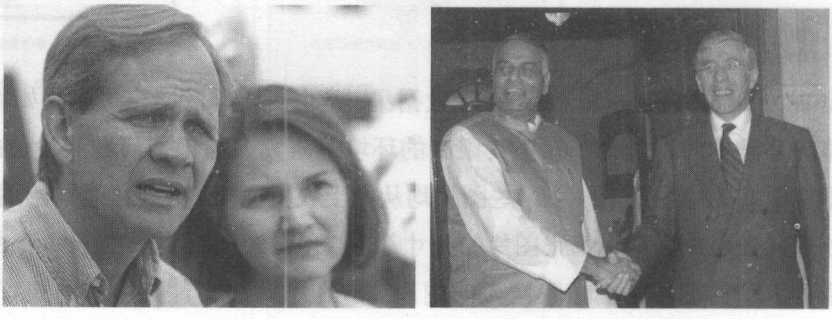


图 1-3　FDDB 数据集中的图像示例

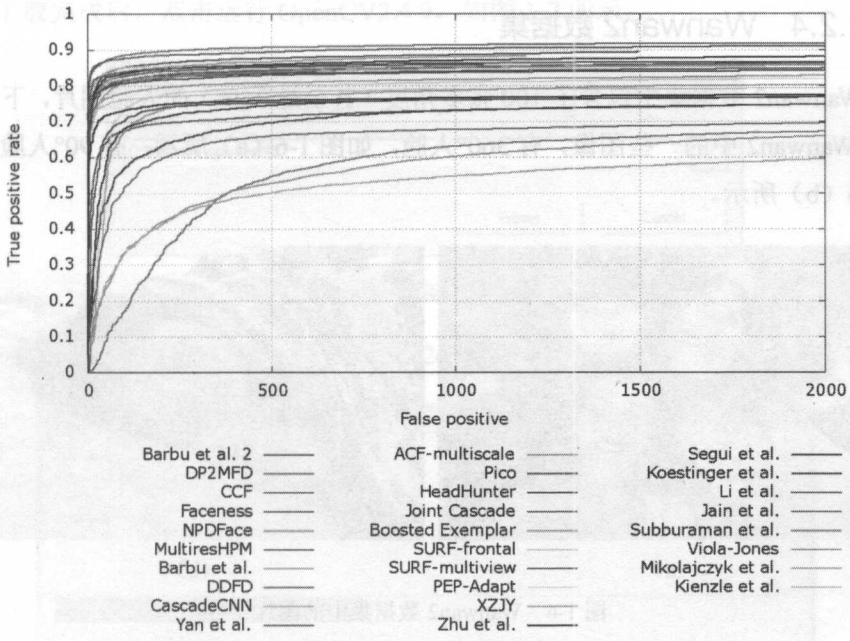



图 1-4　不同人脸检测算法在 FDDB 数据集上检测准确度的 ROC 曲线[3]

### 1.2.3　Wanwan1 数据集

Wanwan1 数据集中包含 100 张侧脸图片，这些图片均来自网上。这些图片尺寸大小不一，场景也不一致。最重要的是，这些图像中的人脸均为侧脸。如图 1-5 所示，呈现了 Wanwan1 数据集中的一些图片。



图 1-5　Wanwan1 数据集中的图片

### 1.2.4 Wanwan2 数据集

Wanwan2 数据集中包含了 100 张多角度（甚至是 360°）的人脸图片，下面呈现了 Wanwan2 中的一些图像，有 360°人脸，如图 1-6（a）所示；有 90°人脸，如图 1-6（b）所示。



（a）　　（b）

图 1-6　Wanwan2 数据集中的图片

## 1.3 OpenCV 的简介、安装与使用

OpenCV 是开源、轻量、运行效率高、最接近商业应用的计算机视觉库，受到工业界和学术界的钟爱。它是以 C/C++实现的，同时提供了 Python、MATLAB、Java、C#等语言的接口。从 2000 年产生至今，OpenCV 已经发布了十多个版本，其功能和算法越来越强大。对于图像处理和计算机视觉领域的初学者和爱好者来说，通过 OpenCV 能够在很短的时间（如几个小时）内快速构建一个可以很好地运行的算法示例，省去了环境配置、内存管理、算法理解、算法实现等耗时、困难的工作。即使领域内的专业人士，也需要经常使用 OpenCV，调用其中的部分关键算法，用于自己的算法研究或实际应用中，因为 OpenCV 的用户量巨大，它所实现的算法一般都是正确的、高效的。下面介绍该软件的安装过程。

#### 1. 下载安装 OpenCV

下载 OpenCV，地址为 http://opencv.org/downloads.html，在这里读者可以看到许多版本的 OpenCV，本书使用的是 Version249 OpenCV for windows。

下载完成后，双击运行 OpenCV2.4.9，如图 1-7 所示。

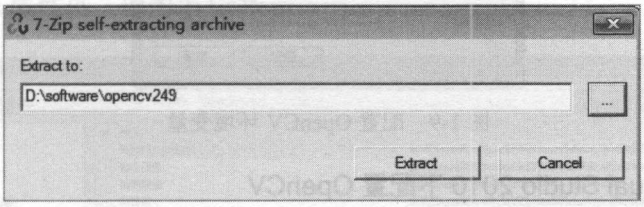



图 1-7　OpenCV 安装时选择提取的路径

选择一个路径，如 D:\software\opencv249，单击“Extract”按钮后如图 1-8 所示。

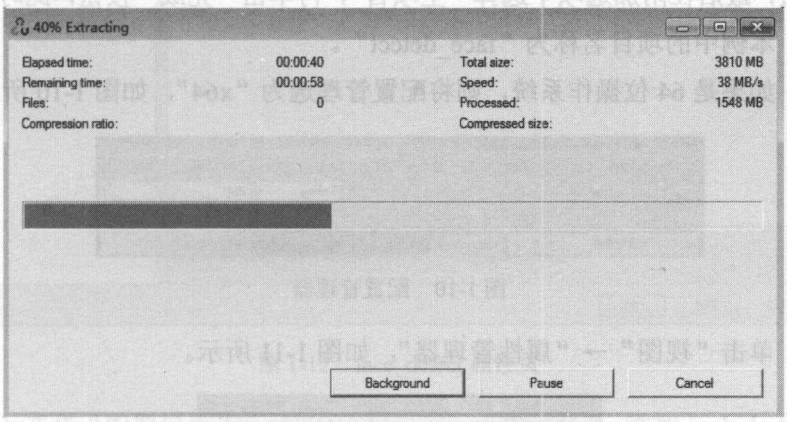



图 1-8　OpenCV 安装进度

等 Extracting 结束后，OpenCV 安装完成。OpenCV 安装在“D:\software\opencv249”文件夹下。

### 2. 配置 OpenCV 的环境变量

选择计算机→（右键）属性→高级系统设置→高级（标签）→环境变量→（双击）Path，在 Path 后面添加相应的路径，如图 1-9 所示。如果是 32 位操作系统，就添加“;… opencv\build\x86\vc10\bin”。如果是 64 位操作系统，就添加“;… opencv\build\x64\vc10\bin”。例如本书中 OpenCV 安装在“D:\software\opencv249”文件夹下，且本书所用的是 64 位操作系统，则在 Path 中添加的路径为：D:\software\opencv249\opencv\build\x64\vc10\bin。




图 1-9　配置 OpenCV 环境变量

### 3. 在 Visual Studio 2010 下配置 OpenCV

（1）创建一个新项目。打开 Visual Studio 2010，单击“文件”→“新建”→“项目”，选择“Win32 控制台应用程序”，输入项目名称和项目路径，单击“下一步”按钮，最后在附加选项中选择“空项目”，再单击“完成”按钮，此时项目创建完成。本例中的项目名称为“face_detect”。

（2）如果是 64 位操作系统，则将配置管理选为“x64”，如图 1-10 所示。




图 1-10　配置管理器

（3）单击“视图”→“属性管理器”，如图 1-11 所示。

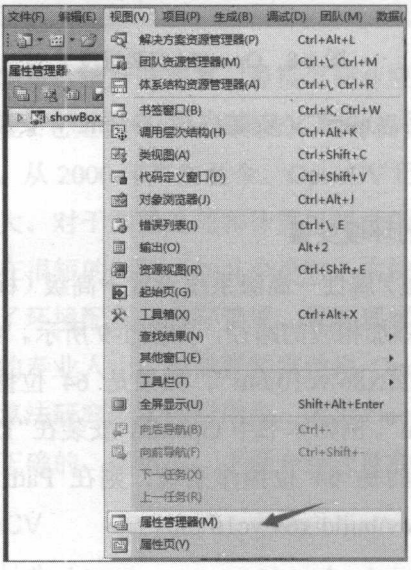



图 1-11　属性管理器

（4）在打开的“属性管理器”工作区中，用鼠标右键单击“face_detect”，打开 face_detect 属性页，如图 1-12 所示。

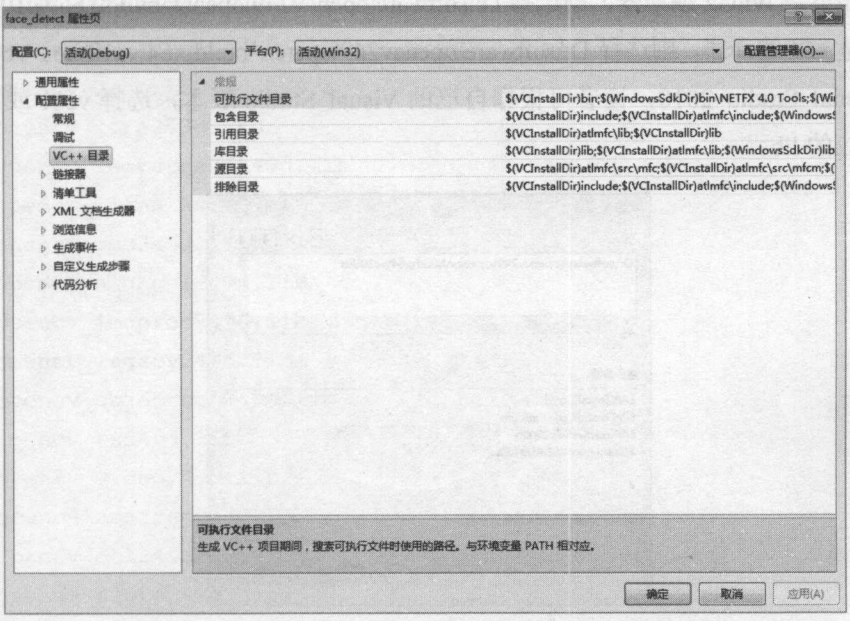



图 1-12　face_detect 属性页

（5）选择“配置属性”→“VC++目录”→“库目录”，添加 include 所在的路径，然后单击“确定”按钮，如图 1-13 所示。

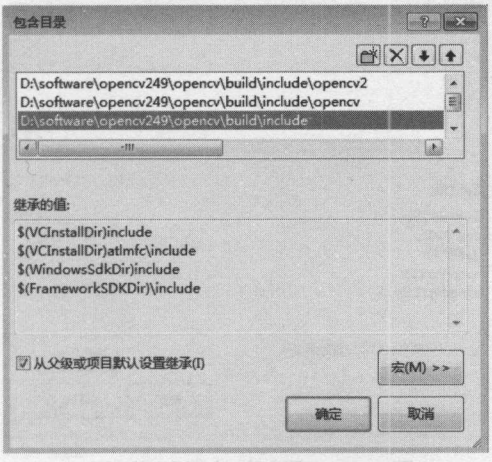



图 1-13　添加 include 路径

（6）选择“VC++目录”下的“库目录”，添加 lib 对应的路径后，单击“确定”按钮，如图 1-14 所示。

如果是 Win32 编译器，则选择 D:\software\opencv249\opencv\build\x86\vc10\lib；如果是 x64 编译器，则选择 D:\software\opencv249\opencv\build\x64\vc10\lib。vc10 代表 Visual Studio 2010，读者可根据自己的 Visual Studio 版本，选择 vc11 或 vc12 下对应的 lib。




图 1-14　添加 lib 路径

（7）单击“连接器”→“输入”→“附加依赖项”，会弹出“附加依赖项”对话框，如图 1-15 所示。

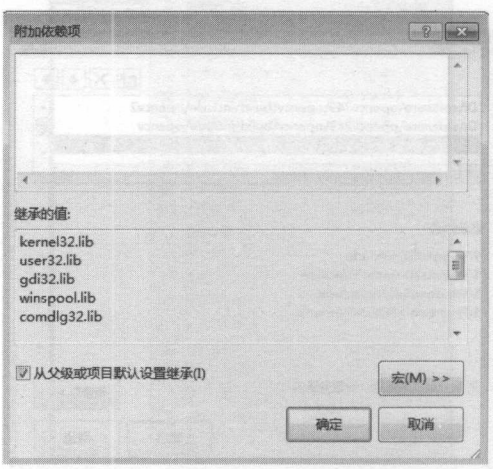



图 1-15 “附加依赖项”对话框

将 D:\software\opencv249\opencv\build\x64\vc10\lib 下的所有文件名，即如下内容添加到附加依赖项中，添加完成后单击“确定”按钮。

```
opencv_ml249d.lib
opencv_calib3d249d.lib
opencv_contrib249d.lib
opencv_core249d.lib
opencv_features2d249d.lib
opencv_flann249d.lib
opencv_gpu249d.lib
opencv_highgui249d.lib
opencv_imgproc249d.lib
opencv_legacy249d.lib
opencv_objdetect249d.lib
opencv_ts249d.lib
opencv_video249d.lib
opencv_nonfree249d.lib
opencv_ocl249d.lib
opencv_photo249d.lib
opencv_stitching249d.lib
opencv_superres249d.lib
opencv_videostab249d.lib
opencv_objdetect249.lib
opencv_ts249.lib
opencv_video249.lib
opencv_nonfree249.lib
opencv_ocl249.lib
opencv_photo249.lib
opencv_stitching249.lib
opencv_superres249.lib
opencv_videostab249.lib
opencv_calib3d249.lib
opencv_contrib249.lib
opencv_core249.lib
opencv_features2d249.lib
opencv_flann249.lib
opencv_gpu249.lib
```

```
opencv_highgui249.lib
opencv_imgproc249.lib
opencv_legacy249.lib
opencv_ml249.lib
```

最后单击“确定”按钮，关闭 face_detect 属性页。此时，Visual Studio 2010 上已经配置好了 OpenCV。

这时，读者可以在源文件中建立一个 CPP 文件，写一个简单的调用 OpenCV 的函数的程序，检验自己的环境是否配置成功。

本书给出了一个 test.cpp，读取图片，并且显示图片。读者可使用 test.cpp 作测试用例。

Code：test.cpp

```
#include<iostream>
#include<string>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

int main(int argc, char* argv[])
{
    string imagename="1.jpg";
    Mat img = imread(imagename);
    if(img.empty())
    {
        fprintf(stderr, "Can not load image %s\n", imagename);
        return -1;
    }

      imshow("image", img); //显示图片

    waitKey();  //等待按键，按任意键即可返回

    return 0;
}
```

运行 test.cpp，结果会显示“1.jpg”。

如果出现“缺少 opencv_*249 文件”之类的错误，则将如下（32 位/64 位操作系统）所有.dll 文件复制到 C:\Windows\System32 下即可。

- 64 位操作系统：D:\software\opencv249\opencv\build\x64\vc10\bin 路径下的所有.dll 文件。
- 32 位操作系统：D:\software\opencv249\opencv\build\x86\vc10\bin 路径下的所有.dll 文件。

三个知名的商业人脸识别软件分别是 OpenCV、Face++、VeriLook，其使用方法将在本书第 8 章讲解。

# 参考文献

[1] Labeled Faces in the Wild (LFW). http://vis-www.cs.umass.edu/lfw/.

[2] Gary B. Huang, Manu Ramesh, Tamara Berg, Erik Learned-Miller. Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. University of Massachusetts, Amherst, Technical Report 07-49, October, 2007.

[3] FDDB: Face Detection Data Set and Benchmark.http://vis-www.cs.umass.edu/fddb/.

[4] Vidit Jain, Erik Learned-Miller. FDDB: A Benchmark for Face Detection in Unconstrained Settings.Technical Report UM-CS-2010-009, Dept. of Computer Science, University of Massachusetts, Amherst. 2010.

[5] Faces in the Wild. http://tamaraberg.com/faceDataset/index.html.

# 第 2 章

# 图像处理基础

本章主要讲解数字图像处理中的基本概念和常见操作。

说到“图像处理”，很多读者都会联想到“图片处理”、“图像的 PS 处理”，即使用图像处理软件（尤其是 Photoshop）对图像进行加工变换，以满足用户的视觉、心理等要求。本书中所述的图像处理，则是指使用图像处理的算法和程序，对图像进行去噪、增强、变换等处理，从图像中提取重要的特征，并结合实际应用对图像进行处理、识别和分析的技术。

## 2.1 数字图像处理的基本概念

数字图像处理（Digital Image Processing）又称为计算机图像处理，它是指将图像信号转换成数字信号并利用计算机对其进行处理的过程。所以计算机处理的图像都是数字化的图像。在 MATLAB 中，图像都是以数值矩阵的形式存在和处理的。

### 2.1.1　像素

像素是构成数字图像的基本单位。一张图片尺寸为 600×300 像素，即表示横向有 600 个像素，纵向有 300 个像素。在数字图像中，每个像素都有对应的数值，二进制图像的像素值为 0 或 1，灰度图像的像素值为 0~255，由这些像素值组成的数字图像才能被计算机进行处理。一个像素的具体长度与显示器的分辨率有关，在 Photoshop 中，1 厘米=28.346 像素，此时 1 像素=0.035 278 厘米。

### 2.1.2　分辨率

分辨率可分为屏幕分辨率（显示分辨率）、图像分辨率、打印分辨率（dpi）三个主要类别。下面主要介绍屏幕分辨率和图像分辨率的概念。

**1. 屏幕分辨率**

屏幕分辨率就是计算机显示器的屏幕上能显示信息的数量，即能显示的像素个数，通常以水平和垂直像素数量来衡量。一台计算机显示器的分辨率为 1600×900 像素，表示该显示器水平方向上能显示 1600 个像素，垂直方向上能显示 900 个像素。对于同一台显示器来说，分辨率越高，能显示的像素就越多，屏幕就越清晰。

查看计算机显示器屏幕分辨率的方式有：（1）通过在计算机（Windows 7 操作系统）的桌面上单击鼠标右键，选择“屏幕分辨率”，如图 2-1 所示。（2）打开“控制面板”，选择“外观和个性化”→“显示”→“调整屏幕分辨率”。调整屏幕分辨率的窗口如图 2-2 所示，可以看到屏幕分辨率有多种选择，如 1600×900、1024×768、800×600 等。通过调整分辨率，观察显示器的清晰程度，可以看到分辨率越高，显示器屏幕越清晰。

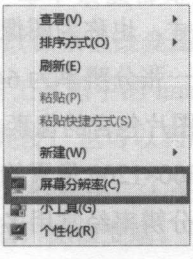



图 2-1　Windows 7 操作系统中查看屏幕分辨率的方法




图 2-2　Windows 7 操作系统中调整屏幕分辨率窗口

在实际应用中，显示设备（如显示器、平板电脑、手机屏幕等）的分辨率也经常用每英寸包含的像素个数（Pixels Per Inch，PPI），即每英寸像素数（像素密度）来表示。该值越大，画面的细节就越丰富；该值越小，图像显示就越大。已知屏幕尺寸和屏幕分辨率时，就可以计算该屏幕的 PPI。例如，一个 5 英寸、分辨率为 1280×720 像素的手机屏幕，其 PPI=sqrt(1280×720/5)=429ppi。类似地，已知屏幕的 PPI 及其尺寸时，也可以计算屏幕分辨率。2010 年，史蒂夫·乔布斯（Steve Jobs）在 iPhone 4 发布会上介绍视网膜技术时，他是这样阐述的：“当你所拿的东西距离你 10～12 英寸（约 25～30 厘米）时，它的分辨率只要在 300ppi 这个‘神奇数字’以上，人眼的视网膜就无法分辨出像素点了。”因此，iPhone 4 屏幕的像素密度就设定在 326ppi。

### 2. 图像分辨率

图像的分辨率就是图像的尺寸，也称为图像的宽和高，即水平方向上的像素个数和垂直方向上的像素个数。一张分辨率为 640×480 像素的图片，有 480 行像素信息，每行有 640 个像素，该图片包含的像素个数是 307 200，也就是我们常说的 30 万像素。而一个分辨率为 1600×1200 像素的图片，就达到了约 200 万的像素。

在实际应用中，视频画面的分辨率经常用多少 P 表示，它表示一个视频画面的垂直像素数，下面将详细介绍。

### 3. 视频分辨率

经常说某个视频是 720P 或 1080P，P 指的是 Progressive，即逐行扫描。480P 是 640×480 像素的分辨率。720P 是由美国电影电视工程协会制定的一种高清电视格式，即 1280×720 像素的分辨率。后来又出现了 1080P 的格式，即分辨率为 1920×1080 像素。现实生活中，不同的视频网站对标清、高清、超清有不同的定义，通常情况下，标清视频画面的分辨率小于等于 480P，而高清视频画面的分辨率为 720P（视频中画面的分辨率为 1280×720 像素），超清视频画面的分辨率则为 1080P（视频中画面的分辨率为 1920×1080 像素）。

## 2.1.3　图像的色调、亮度和饱和度

### 1. 图像的色调

色调就是通常意义下的彩色，它随波长的变化而变化，反映理论颜色的基本特征。在三原色中，可见光的波长为 380～780nm（纳米），不同波长呈现出不同的颜色，可见波长从长到短依次为红、橙、黄、绿、青、蓝、紫。红色的波长为 700nm、绿色的波长为 546.1nm、蓝色的波长为 435.8nm。只有单一波长成分的光称为单色光，含有两种以上波长成分的光称为复合光。色彩通常都是由一种单色光和白光按照一定比例混色的。色调和饱和度合称为色度，色度表示的是色彩的纯度。色度反映颜色的色调和饱和度。颜色由亮度和色度共同表示。

### 2. 图像的亮度

亮度是人眼对光强度的感受，图像的亮度就是图像中光的强度。不同波长的光有着不同的色调，给人不同的颜色感受，也给人不同的亮度感受。人眼一般感到红光最暗，蓝光次之，而黄绿光最亮。

数字图像由数值矩阵表示，每个数值都是图像的像素值。下面我们借助 MATLAB 工具，介绍如何通过修改像素值改变图像的亮度。

我们使用 MATLAB 图像处理工具箱中的 imadd()函数，让数字图像的像素值增加一个常数，从而改变图像的亮度；也可以利用 MATLAB 图像处理工具箱中的 immultiply()函数，让数字图像矩阵乘以一个常数，如果常数大于 1，则图像的亮度增强，否则图像的亮度将降低，图像变暗。下面我们通过 Code1 介绍如何利用这些函数改变图像的亮度。

Code1：ch2_1.m

```
close all;clear all; clc; %关闭图形窗口，清除工作空间变量，清空命令行
img=imread('1.jpg'); %读图片
img1=imadd(img,100); %在图像上增加一个常数
img2=immultiply(img,2); %在图像上乘以一个大于 1 的常数
img3=immultiply(img,0.5);  %在图像上乘以一个小于 1 的常数
%分别显示原图、增加常数、乘以常数后的图像
figure,
subplot(2,2,1),imshow(img),xlabel('(a)原图像 ');
subplot(2,2,2),imshow(img1),xlabel('(b)增加一个常数后的图像');
subplot(2,2,3),imshow(img2),xlabel('(c)乘以大于 1 的常数后的图像');
subplot(2,2,4),imshow(img3),xlabel('(d)乘以小于 1 的常数后的图像');
```

运行 Code1 后，结果如图 2-3 所示。我们可以看到，处理后的图像（b）、（c）和原图像（a）相比，亮度明显增加；图像（d）的亮度和原图像（a）相比，明显降低。



（a）原图像



（b）增加一个常数后的图像



（c）乘以大于 1 的常数后的图像



（d）乘以小于 1 的常数后的图像

图 2-3　不同图像亮度的试验结果

此外，还可以利用 MATLAB 中的其他函数改变图像的亮度，如 imadjust()函数（调整参数 Gamma，如果 Gamma 小于 1，则图像亮度增加；如果 Gamma 大于 1，则图像亮度变暗）、brighten()函数（调整参数 Beta，如果 0<Beta<1，则图像亮度增加；如果–1<Beta<0，则图像亮度变暗）。读者可以查阅 MATLAB 帮助文档进行这些函数的学习。

### 3. 图像的饱和度

色彩的饱和度就是颜色的纯净度，饱和度越高，色彩越鲜明。色彩的饱和度取决于颜色中白光的比例，色彩中的白光越多，饱和度就越低。纯单色光的饱和度为 100%，纯白光的饱和度为 0%。

下面我们通过使用 Photoshop 调整一幅图像的饱和度，观察调整前后图像的变化。具体步骤如下：

（1）在 Photoshop 中打开一幅图像，如图 2-4（a）所示。

（2）选择“图像”→“调整”→“色相/饱和度”，打开“色相/饱和度”对话框，如图 2-5 所示。

（3）调整饱和度的值。将饱和度调整为 100，对应的图片如图 2-4（b）所示；将饱和度调整为–100，对应的图片如图 2-4（c）所示。



（a）原图像　　（b）饱和度为 100　　（c）饱和度为–100

图 2-4　同一图片不同饱和度值的变化

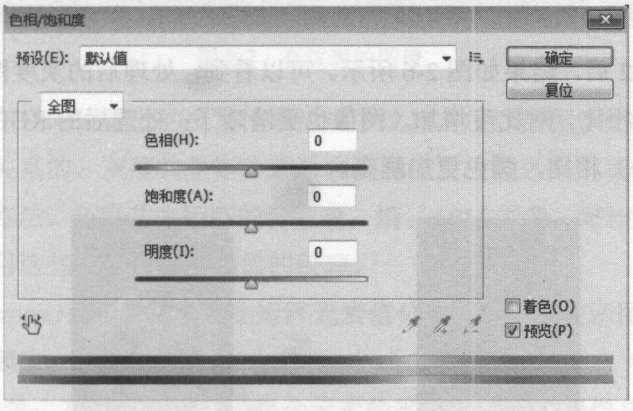



图 2-5　“色相/饱和度”对话框

上面已经提到，色调和饱和度合称为色度。亮度和色度（色调和饱和度）共同决定了图像的特性。

### 2.1.4 图像的对比度

对比度是图像的黑白之间的差异，差异越大，则对比度越大，能表示的颜色就越丰富。但是现在还没有通用标准来衡量对比度，需要依靠人的眼睛进行判断。

在 MATLAB 中，函数 imadjust()可以增加图片的对比度，格式如下。

```
Newimg=imadjust(img);
```

下面我们通过 Code2 了解如何利用函数 imadjust()增加图片的对比度。

Code2：ch2_2.m

```
close all;clear all;clc;
img=imread('pout.tif');  %读取灰度图像
newimg=imadjust(img); %增加对比度
img1=imread('1.jpg'); %读取 RGB 图像
newimg1=imadjust(img1,[0.1 0.2 0; 0.5 0.6 1],[] );  %增加对比度
figure,
subplot(1,2,1);imshow(uint8(img)); xlabel('(a)原灰度图像');
subplot(1,2,2);imshow(uint8(newimg));xlabel('(b)修改对比度后的图像');
figure,
subplot(1,2,1);imshow(uint8(img1)); xlabel('(a)原 RGB 图像');
subplot(1,2,2);imshow(uint8(newimg1));xlabel('(b)修改对比度后的图像');
```

运行 Code2 后，结果如图 2-6 所示。可以看到，处理后的灰度图像（b）和对应的原图（a）相比，对比度增加，图像也更清晰了；处理后的 RGB 图像（d）和对应的原图（c）相比，颜色更加鲜亮。



（a）原灰度图像



（b）修改对比度后的图像

图 2-6 Code2 的运行结果




（c）原 RGB 图像　　（d）修改对比度后的 RGB 图像

图 2-6　Code2 的运行结果（续）

## 2.1.5　图像的纹理

图像的纹理特征是描述一幅图像的重要性质之一，图像的纹理、颜色和形状一起用来表述图像的视觉特征，也就是人的视觉能感受到的自然特征。

纹理，通常意义上指的是物体表面凹凸不平的沟纹，或者光滑物体表面的彩色图案，这些图案会让人有凹凸不平的视觉感受。图 2-7 所示的三幅图像就是有纹理的图像。

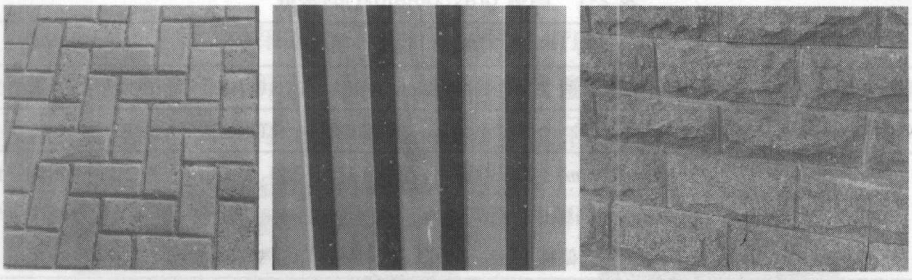


图 2-7　有纹理的图像

灰度分布在空间位置上的反复出现形成了纹理，所以图像空间中一定距离的两个像素是有关系的，灰度共生矩阵就是利用图像灰度级之间的相关性来描述纹理的一种常用方法。灰度共生矩阵的特征有：熵、ASM 能量、对比度、均匀度和自相关，可以用这些特征来分析图像的纹理。

- 熵（Entropy）：如果灰度共生矩阵元素值分布均匀，则表示图像近于随机，熵值就越大。
- ASM 能量：灰度共生矩阵中每个元素的平方和，表示图像是否分布均匀、纹理是否粗糙。图像越有结构，ASM 越大。

- 对比度（Contrast）：表示图像的清晰程度和纹理深浅程度。灰度共生矩阵中偏离对角线的元素值越大，对比度就越大，纹理也就越深。
- 均匀度：表示图像纹理的粗细度。纹理越粗糙，则均匀度越大；纹理越细致，则均匀度越小。
- 自相关（Correlation）：描述灰度共生矩阵中水平或垂直方向上的相似度。如果矩阵中水平方向的元素值均匀相等，则水平方向的相关性大，对应的图像在水平方向上越有纹理，则水平方向灰度共生矩阵的自相关值大于其他方向灰度共生矩阵的自相关值。

在 MATLAB 中，函数 graycomatrix()可以生成一幅图像的灰度共生矩阵，格式如下。

```
glcm=graycomatrix(img,[param1,val1,param2,val2…])
```

img 是输入的原图像，glcm 是返回的灰度共生矩阵。函数 graycomatrix()可以设置一些参数，如表 2-1 所示。

表 2-1　函数 graycomatrix()的参数

| Param | 说　明 |
|---|---|
| GrayLimits | 对应的 value 是：两个元素的向量[low,high]，如[1,100]。表示灰度映射范围 |
| NumLevels | 对应的 value 是：一个数值，如 8。表示图像归一化的范围 |
| Offset | 对应的 value 是：$m$×2 的二维数组，如{[0,2]、[-2,2]、[-2,0]、[-2,-2]}。表示偏移角度和距离。[0,2]为偏移 0°，距离为 2；[-2,2]为偏移 45°，距离为 2；[-2,0]为偏移 90°，距离为 2；[-2,-2]为偏移 135°，距离为 2 |

在 MATLAB 中，函数 graycoprops()可以获取一个共生矩阵的熵、ASM 能量、对比度和自相关的属性值，格式如下。

```
stats=graycoprops(glcm,{'contrast','correlation','energy','homo
geneity'});
```

下面我们通过 Code3 说明如何利用函数 graycomatrix()生成图像对应的灰度共生矩阵，并利用函数 graycoprops()获取共生矩阵的属性值。

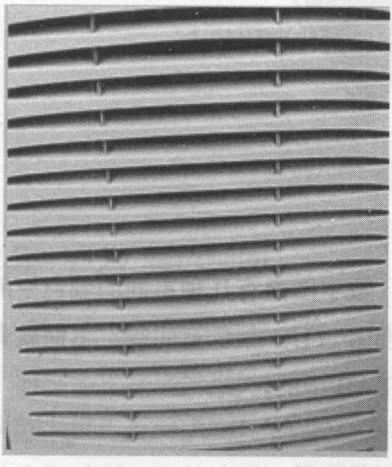
Code3：ch2_3.m

```
close all;clear all;clc;
img1=imread('kt.jpg'); %读取 RGB 类型的纹理图像(a)
img1=rgb2gray(img1);  %将 RGB 图像转换为灰度图像
```

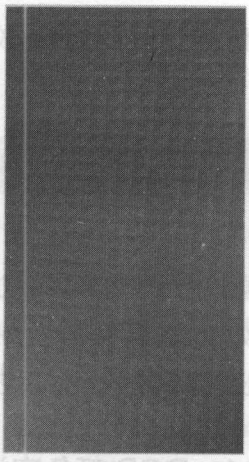
```
    glcm1=graycomatrix(img1,'Offset',[0 2]); %得到图像的灰度共生矩阵
glcm1
    %显示灰度图像 img1
    figure,
    imshow(img1);
    %获取 glcm1 的对比度、自相关、熵的属性值
    stats1=graycoprops(glcm1,{'contrast','correlation','energy'});

    img2=imread('dn.jpg'); %读取 RGB 类型的纹理图像(b)
    img2=rgb2gray(img2); %将 RGB 图像转换为灰度图像
    glcm2=graycomatrix(img2,'Offset',[0 2]); %得到图像的灰度共生矩阵
glcm2
    %显示灰度图像 img2
    figure,
    imshow(img2);
    %获取 glcm2 的对比度、自相关、熵的属性值
    stats2=graycoprops(glcm2,{'contrast','correlation','energy'});
```

运行 Code3 后，结果如图 2-8 所示。在 MATLAB 工作空间中查看 glcm1 和 glcm2 的值，如图 2-9 所示。最后，统计两幅纹理图像的对比度、自相关、熵的属性值，即工作空间中的 stats1 和 stats2，结果如表 2-2 所示。



（a）空调风格栅图



（b）电脑外壳图

图 2-8　Code3 的运行结果

glcm1

glcm1 <8x8 double>

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 4148 | 344 | 40 | 29 | 23 | 14 | 10 | 0 |
| 2 | 325 | 1702 | 449 | 42 | 22 | 21 | 12 | 1 |
| 3 | 49 | 410 | 1802 | 457 | 43 | 30 | 9 | 0 |
| 4 | 31 | 55 | 434 | 1394 | 442 | 50 | 11 | 0 |
| 5 | 23 | 23 | 48 | 420 | 1784 | 542 | 22 | 0 |
| 6 | 12 | 17 | 19 | 55 | 559 | 4039 | 674 | 9 |
| 7 | 9 | 6 | 3 | 12 | 39 | 810 | 16725 | 868 |
| 8 | 0 | 3 | 0 | 0 | 0 | 9 | 897 | 5118 |

glcm2

glcm2 <8x8 double>

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 16335 | 3522 | 0 | 0 | 0 |
| 4 | 0 | 0 | 3267 | 58919 | 821 | 0 | 3 |
| 5 | 0 | 0 | 0 | 990 | 2070 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 2 | 1 | 0 | 0 |
| 8 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

图 2-9　两幅纹理图像对应的灰度共生矩阵

**表 2-2　两幅纹理图像对应的灰度共生矩阵的属性值**

| 属性 / 图像 | 对比度 | 自相关 | 熵 |
|---|---|---|---|
| 空调风格栅图 | 0.3219 | 0.9682 | 0.1744 |
| 电脑外壳图 | 0.1010 | 0.7773 | 0.5102 |

空调风格栅图的纹理比较深，所以它的对比度比较大。

## 2.2　颜色空间

根据数字图像色彩表示方式的不同，数字图像有几种不同的颜色空间（颜色模型），如 RGB 颜色空间、HSV 颜色空间、YUV 颜色空间等。下面就具体介绍一下这几种颜色空间，以及它们在 MATLAB 中的转换。

### 2.2.1　RGB 颜色空间

RGB 颜色空间是由 Red（红色）、Green（绿色）、Blue（蓝色）组成的一种颜

色空间。红、绿、蓝这三种颜色被国际照明委员会规定为三原色，这三原色的不同分量比例可以合成任何其他颜色。RGB 颜色空间类似一个三维的坐标系，红色、绿色和蓝色分别是一个坐标轴，每种颜色都是空间中的点，由三个数值组成，分别表示在红色、绿色、蓝色上的分量。(0,0,0)表示纯黑色，(255,255,255)表示纯白色，(0,0,255)表示蓝色。

## 2.2.2　HSV 颜色空间

HSV 颜色空间由 Hue（色调）、Saturation（饱和度）和 Value（亮度）组成。HSV 颜色空间是 1978 年提出的，它是一种主观的颜色空间。HSV 颜色空间类似一个六角椎体，如图 2-10 所示。色调通常指代颜色名称。饱和度表示掺入白光的分量，掺入白光的分量越多，则饱和度越低，即 S 值越小；掺入白光的分量越少，则饱和度越高，即 S 值越大。亮度表示掺入黑光的分量，掺入黑光的分量越多，则亮度越低，即 V 越小；掺入黑光的分量越少，则亮度越高，即 V 越大。

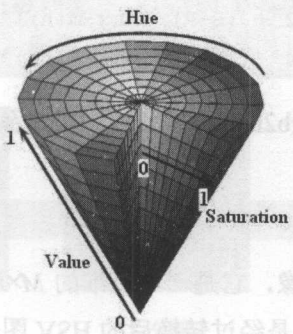



图 2-10　HSV 颜色空间模型

## 2.2.3　YUV 颜色空间

YUV 颜色空间是欧洲电视系统使用的一种色彩编码空间，在现代的彩色电视系统中，通过三管彩色摄影机和彩色 CCD 摄影机得到的彩色图像信号，经过分色、分别放大校正得到 RGB，再进一步经矩阵变换电路得到亮度信号 Y 和两个色差信号 R-Y（U）、B-Y（V），最后发送端对这三个信号分别编码，再使用同一信道发出去，这就是 YUV 颜色空间。若只有亮度 Y，没有 U 和 V，则形成了黑白灰度图像。

在数字图像中，图像的色彩空间还有很多种，如 CMYK、Lab、HSL、YIQ 等，本书在此不多做介绍。

### 2.2.4 颜色空间的转换

#### 1. RGB 颜色空间转换为 HSV 颜色空间

RGB 颜色空间中有三个分量，即 R、G、B。HSV 颜色空间中也有三个分量，分别是 H、S、V。可使用如下算法将 RGB 颜色空间转换为 HSV 颜色空间。

```
%RGB 到 HSV 的转换算法
max=max(R,G,B)
min=min(R,G,B)
S=(max-min)/255; V=max;
if(max==min),H=0,S=0;
else
    if R = max, H = (G-B)/(max-min)*60;
    if G = max, H = (2 + (B-R)/(max-min))*60
    if B = max, H = (4 + (R-G)/(max-min))*60
```

在 MATLAB 中，函数 rgb2hsv()可以实现 RGB 到 HSV 颜色空间的转换，用法如下。

```
hsv=rgb2hsv(rgb)
```

rgb 是输入的原 RGB 图像，它是一个三维的 $M\times N\times 3$ 的数组，分别是图像的红色、绿色和蓝色的分量。hsv 是经过转换后的 HSV 图像，它是一个三维的 $M\times N\times 3$ 的数组，分别表示图像的色调、饱和度和亮度信息。

下面我们通过 Code4 了解如何利用函数 rgb2hsv()进行 RGB 到 HSV 颜色空间的转换。

Code4：ch2_4.m

```
close all;clear all;clc;
img=imread('1.jpg'); %读取 RGB 图像
hsv=rgb2hsv(img);    %转换为 HSV 颜色空间
figure,
subplot(1,2,1);imshow(img);xlabel('(a)原图像')
```

```
subplot(1,2,2);imshow(hsv);xlabel('(b)转换后的 HSV 图像')
figure ,
subplot(1,3,1);imshow(hsv(:,:,1));xlabel('H 分量');
subplot(1,3,2);imshow(hsv(:,:,2));xlabel('S 分量');
subplot(1,3,3);imshow(hsv(:,:,3));xlabel('V 分量');
```

运行 Code4 后，结果如图 2-11 所示，有两组图，即 Figure1 和 Figure2。Figure1 是原 RGB 图像和转换后的 HSV 图像的对比，Figure2 是转换后的 HSV 图像的色度、饱和度和亮度分量图。



（a）原 RGB 图像　　（b）转换后的 HSV 图像

Figure1：原 RGB 图像和转换后的 HSV 图像



H 分量　　S 分量　　V 分量

Figure2：转换后的 HSV 图像的 H、S、V 分量图

图 2-11　Code4 的运行结果

### 2. HSV 颜色空间转换为 RGB 颜色空间

在 MATLAB 中，也有和 rgb2hsv 类似的函数。函数 hsv2rgb()就可以实现 HSV 到 RGB 颜色空间的转换，格式如下。

```
rgb=hsv2rgb(hsv)
```

hsv 是输入的原 HSV 图像，rgb 是经过转换后的 RGB 图像。

下面我们借助 Code5 了解如何利用函数 hsv2rgb()进行 HSV 到 RGB 颜色空间的转换。

Code5：ch2_5.m

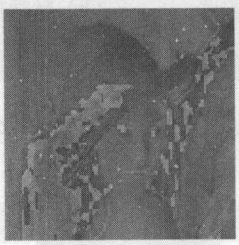
```
close all;clear all;clc;
img=imread('1.jpg'); %读取 RGB 图像
hsv=rgb2hsv(img);   %转换为 HSV 颜色空间
newrgb=hsv2rgb(hsv);  %将 HSV 图像转换为 RGB 图像
figure,
subplot(1,3,1);imshow(img);xlabel('(a)原图像')
subplot(1,3,2);imshow(hsv);xlabel('(b)转换后的 HSV 图像')
subplot(1,3,3);imshow(newrgb);xlabel('(c)转换后的 RGB 图像')
```

程序 Code5 首先读取 RGB 图像“1.jpg”，利用函数 rgb2hsv()将 RGB 图像转换为 HSV 图像，然后再利用函数 hsv2rgb()将上一步得到的 HSV 图像转换为一个新的 RGB 图像。最后显示原 RGB 图像、转换后的 HSV 图像和转换后的新 RGB 图像。

运行 Code5 后，结果如图 2-12 所示。可以看到，转换后的 RGB 图像（c）和原 RGB 图像（a）是完全相同的。



（a）原 RGB 图像



（b）转换后的 HSV 图像



（c）转换后的 RGB 图像

图 2-12　Code5 的运行结果

### 3．RGB 颜色空间转换为 YUV 颜色空间

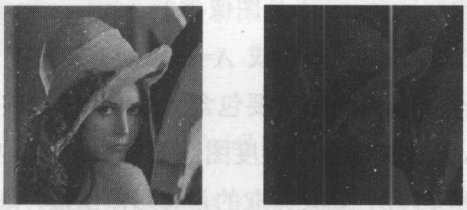
在 MATLAB 中，没有用于将 RGB 颜色空间转换为 YUV 颜色空间的函数。Code6 程序可以实现 RGB 到 YUV 颜色空间的转换。

Code6：ch2_6.m

```
close all;clear all;clc;
rgb=imread('1.jpg');  %读取 RGB 图像
r=rgb(:,:,1);
g=rgb(:,:,2);
b=rgb(:,:,3);
```

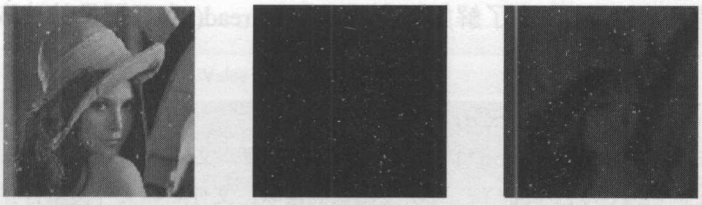
```
rgb=double(rgb); %rgb 变为 double 类型
%RGB 到 YUV 颜色空间的转换
y =(0.299 * r + 0.587 * g + 0.114 * b);
u = ((b - y) * 0.492);
v = ((r - y) * 0.877);
yuv(:,:,1)=y;
yuv(:,:,2)=u;
yuv(:,:,3)=v;
figure,
subplot('1,2,1');imshow(uint8(rgb)); xlabel('(a)原 RGB 图像');
subplot('1,2,2');imshow(uint8(yuv)); xlabel('(b)转换后的 YUV 图像');
figure,
subplot('1,3,1');imshow(y);xlabel('(a)转换后的 Y 分量');
subplot('1,3,2');imshow(u);xlabel('(b)转换后的 U 分量');
subplot('1,3,3');imshow(v);xlabel('(c)转换后的 V 分量');
```

运行 Code6 后，结果如图 2-13 所示，有两组图，即 Figure1 和 Figure2。Figure1 是原 RGB 图像和转换后的 YUV 图像的对比，Figure2 是转换后的 YUV 图像的亮度信号 Y 和两个色差信号 U、V 的分量图。



（a）原 RGB 图像　（b）转换后的 YUV 图像

Figure1：原 RGB 图像和转换后的 YUV 图像



（a）转化后的 Y 分量　（b）转化后的 U 分量　（c）转化后的 V 分量

Figure2：转换后的 YUV 图像的 Y、U、V 分量图

图 2-13　Code6 的运行结果

## 2.3 数字图像处理的基本操作

在数字图像处理中，计算机对图像有着很多处理和操作方法，其中最常见的是图像的读取、显示、修改和保存，以及获取图像的基本信息。本节会对这些内容进行介绍，并结合相关内容对图像进行一些简单的处理。

### 2.3.1 图像的读取

如果要对数字图像进行处理，则首先需要读取图片。在 MATLAB 中，函数 imread()可以实现对图像的读取，其格式如下。

灰度图像或真彩色图像的读取：

```
A = imread('filename','fmt')或imread('filename .fmt')
```

索引图像的读取：

```
[X,map] = imread('filename' , 'fmt')或imread('filename .fmt')
```

filename 是图像的文件名，fmt 是图像类型（如.jpg、.gif、.jpeg、.png 等）。图像读取可以为 A=imread('lena','jpg')或 A=imread('lena.jpg')。如果 lena.jpg 不在 MATLAB 的当前路径下，则 filename 要包含文件的路径。该函数读取的图像为灰度图像或真彩色图像，如果读取的是灰度图像，则 A 是一个 $M\times N$ 的二维数组（$M$、$N$ 分别是图像的高度和宽度）；如果读取的是真彩色图像，则 A 是一个 $M\times N\times 3$ 的三维数组。如果读取的是索引图像，则 X 代表数据矩阵，是 $M\times N$ 的二维数组；map 代表调色板矩阵，是 $m\times 3$ 的二维数组。

下面我们通过 Code7 来了解如何利用函数 imread()实现图像的读取。

Code7：ch2_7.m

```
close all;clear all;clc;
gray1=imread('yy','jpg'); %灰度图像的读取，文件名和文件类型分开
gray2=imread('yy.jpg');  %灰度图像的读取，文件名和文件类型合并
rgb1=imread('1.jpg');  %真彩色图像的读取，图像在当前路径下
rgb2=imread('E:\five.jpg'); %真彩色图像的读取，图像不在当前路径下，需写完整路径
```

```
figure,
subplot(2,2,1),imshow(gray1),xlabel('gray1');
subplot(2,2,2),imshow(gray2),xlabel('gray2');
subplot(2,2,3),imshow(rgb1),xlabel('rgb1');
subplot(2,2,4),imshow(rgb2),xlabel('rbg2');
```

上述代码演示了读取图像的不同方式，gray1、gray2 读取的是灰度图像，rgb1、rgb2 读取的是真彩色图像。运行 Code7 后，结果如图 2-14 所示。



gray1



gray2



rgb1



rgb2

图 2-14　Code7 的运行结果

Code7 运行结束后，查看 MATLAB 的工作空间，可以看到四个矩阵，即 gray1 和 gray2 是灰度图像，对应的是二维数组；rgb1 和 rgb2 是真彩色图像，对应的是三维数组，如图 2-15 所示。

Workspace

| Name | Value | Min |
|---|---|---|
| gray1 | <400x300 uint8> | 0 |
| gray2 | <400x300 uint8> | 0 |
| rgb1 | <220x220x3 uint8> | 0 |
| rgb2 | <827x1024x3 uint8> | <Too.. |

图 2-15　Code7 运行结束后 MATLAB 的工作空间

函数 imread()还可以读取特殊格式的图像文件。关于函数 imread()特殊使用和图像读取的更多问题，读者可参阅 MATLAB 帮助文档。

### 2.3.2 图像的显示

通常对一幅图像处理完成后，还需要将图像显示出来，查看处理后的效果。在 MATLAB 中，函数 imshow()可以实现对图像的显示。

```
imshow('img')
```

img 可以是 imread()返回的变量，也可以为图片路径，如“E:\five.jpg”。

使用函数 imshow()，MATLAB 会自动打开图形图像视窗，这是 MATLAB 中一种通用的图像显示方法，可以显示灰度图像、真彩色图像、二值图像、索引图像等。在显示灰度图像时，可以加入一些限制，如 imshow(gray,[40,120])，像素值小于 40 的会显示为黑点，像素值大于 120 的会显示为白点。下面我们通过 Code8 来说明如何利用函数 imshow()实现图像的显示。

Code8：ch2_8.m

```
close all;clear all;clc;
gray1=imread('yy.jpg'); %读取灰度图像
rgb1=imread('1.jpg');  %读取 RGB 图像
figure,
subplot(2,2,1),imshow(gray1),xlabel('(a)gray1 读取后显示');
subplot(2,2,2),imshow(rgb1),xlabel('(b)rgb1 读取后显示');
subplot(2,2,3),imshow(gray1,[40,120]),xlabel('(c)限制灰度图像像素值范围');
subplot(2,2,4),imshow('E:\five.jpg'),xlabel('(d)直接显示图像文件');
```

运行 Code8 后，结果如图 2-16 所示。图（a）是对读取的灰度图像进行显示；图（b）是对读取的 RGB 图像进行显示；图（c）在显示时限制了灰度图像（a）的像素值范围；图（d）是根据路径直接显示图像文件。




图 2-16　Code8 运行后的结果

注意：图像显示是先打开图形图像窗口，然后将图片显示在窗口内，如图 2-16 所示。为了清晰展示运行结果，本书中其他图例只显示图像部分，不显示整个图形图像窗口。

在 MATLAB 中，图像的显示函数还有 imtool()、image()、imagesc()、colorbar()、montage()等，关于这些函数的使用，读者可查阅 MATLAB 帮助文档。

### 2.3.3　图像的修改

在数字图像处理中，通常需要对图像进行修改。修改图像就是修改对应的数字矩阵中的像素值。下面我们结合程序 Code9 对图像的修改进行介绍。

Code9：ch2_9.m

```
close all;clear all;clc;
img=imread('yy.jpg'); %读取图片
newimg=img; %将原图像赋值为 newimg，在 newimg 上做修改
newimg(1:100,1:100)=0;%将新图片中的前 100 行、100 列区域内的像素值修改为 0，即变为黑色
figure,
subplot(1,2,1);imshow(img); xlabel('(a)原图像')
subplot(1,2,2);imshow(newimg); xlabel('(b)修改后的图像')
```

运行 Code9 后，结果如图 2-17 所示。我们可以看到，修改后的图像的左上角区域为黑色，因为图像该区域内的像素值被修改为 0。



（a）原图像　　（b）修改后的图像

图 2-17　Code9 的运行结果

### 2.3.4　图像的保存

对数字图像处理结束后，通常需要对处理后的图像进行保存。在 MATLAB 中，函数 imwrite()可以实现图像的保存，格式如下。

```
imwrite(img,filename,fmt)
```

img 是函数 imread()得到的图像矩阵，filename 和 fmt 分别是要存储的文件名和文件类型。如果要保存的 img 是灰度图像，则 img 是一个二维矩阵；如果要保存的 img 是真彩色图像，则 img 是一个三维矩阵。下面我们通过 Code10 说明如何利用函数 imwrite()实现图像的保存。

Code10：ch2_10.m

```
close all;clear all;clc;
gray1=imread('yy.jpg'); %读取灰度图像
rgb1=imread('1.jpg');  %读取真彩色图像
imwrite(gray1,'gray.jpg'); %将灰度图像保存在当前路径下
imwrite(rgb1,'E:\rgb.jpg');  %将 RGB 图像保存在指定的路径下
```

首先读入了一个灰度图像 gray1，然后读入了一个真彩色图像 rgb1，可以在工作空间中看到 gray1 是二维矩阵，rbg1 是三维矩阵。通过函数 imwrite()把图像矩阵保存在指定路径下。运行 Code10 后，当前路径下增加了一个灰度图像，名字为 gray.jpg；计算机 E:盘下增加了一个真彩色图像，名字为 rgb.jpg。

通过函数 imshow()显示图像后，图像会显示在图形图像视窗内，如果利用 hode on 在该图像上添加一个矩形框生成新图像，由于图像的像素值没有改变，不

能使用函数 imwrite()保存新图像，那么此时需要使用函数 saveas()，格式如下。

```
saveas(gcf,'filename','fmt')
```

saveas 是将图形图像窗口保存，相当于将该窗口进行另存为。

下面我们通过 Code11 说明如何利用函数 saveas()保存图像。

Code11：ch2_11.m

```
close all;clear all;clc;
rgb1=imread('1.jpg'); %读取图像
figure, %打开图形图像窗口
imshow(rgb1);  %显示图片
hold on;  %在图像上绘制另一个图像
hr = rectangle('Position', [100,100,80,80]); %绘制的为矩形框
set(hr, 'EdgeColor', [1 0 0]);  %设置矩形框颜色
set(hr, 'LineWidth', 3);  %设置矩形框线的粗细
saveas(gcf, '1_1.jpg');  %保存该图形图像窗口
```

运行 Code11 后，结果如图 2-18 所示。



图 2-18　Code11 的运行结果

## 2.3.5　获取图像的基本信息

在数字图像处理中，通常需要查看图像的基本信息，如图像格式、图像大小、图像数据类型等。在 MATLAB 中，函数 imfinfo()可以获取图像的基本信息，格式如下。

```
INFO=imfinfo(imagename);
```

INFO 返回图像的基本信息，它是一个结构数组。

通过在 MATLAB 的命令行窗口中输入如下语句，来查看图像的基本信息。

```
>>info=imfinfo('1.jpg');
>>info
info =

          Filename: 'E:\wwwang\写书\ch1\1.jpg'
       FileModDate: '07-Jan-2016 15:58:15'
          FileSize: 7927
            Format: 'jpg'
     FormatVersion: ''
             Width: 220
            Height: 220
          BitDepth: 24
         ColorType: 'truecolor'
   FormatSignature: ''
   NumberOfSamples: 3
      CodingMethod: 'Huffman'
     CodingProcess: 'Sequential'
           Comment: {}
```

返回的 info 包含着图像“1.jpg”的基本信息。Filename 代表图像文件的路径和名字；FileModDate 表示图像文件的最后修改时间；FileSize 表示图像文件的大小；Format 表示图像的格式；Width 表示图像的宽；Height 表示图像的高；BitDepth：24；ColorType 表示图像类型（灰度图像、RGB 图像、索引图像、二值图像）。

如果仅需要获取图像的行和列，则可以使用如下语句。

```
row=size(img,1) %得到图像的行
col=size(img,2) %得到图像的列
```

## 2.4 图像类型及转换

根据图像在计算机中表示形式的不同，形成了几种不同类型的图像，如 RGB 图像、灰度图像、索引图像、二值图像等。本节将分别介绍这 4 种图像类型，以及它们之间的相互转换。

## 2.4.1　图像类型

### 1．RGB 图像

RGB 图像，由 Red、Green、Blue 三种颜色分量组成。这三种颜色的组合可形成其他颜色，所以用它们来表示彩色图像。RGB 图像也称真彩色图像，由 3 个维度相同的二维数组组成，每个数组分别代表图像颜色中红色的分量、绿色的分量和蓝色的分量。每个像素由 24 位组成，每 8 位代表一种颜色分量。通过函数 imread()读入一幅真彩色图像，返回的是一个 $M \times N \times 3$ 的三维数组。

### 2．灰度图像

灰度图像的像素值都在[0,255]，一幅灰度图像由一个二维矩阵表示，像素值 0 表示黑色，像素值 255 表示白色。像素值为[1,254]，则表示不同程度的灰色。

### 3．索引图像

索引图像由数据矩阵和调色板矩阵组成。调色板矩阵是一个 $M \times 3$ 的数组，每一行代表一种颜色，每种颜色由 Red、Green、Blue 三种颜色分量组成，颜色分量值为[0,1]的浮点型数据。数据矩阵是一个二维矩阵，数据矩阵中的值指向调色板对应行的数据，如值 1 指向调色板矩阵的第一行，值 5 指向调色板矩阵的第五行。

数据矩阵和调色板矩阵存放在一起，通过函数 imread()读取一幅索引图像，会返回数据矩阵 X 和调色板矩阵 map，格式如下。

```
[X,map]=imread('索引图像');
```

### 4．二值图像

二值图像是由像素值 0 或 1 组成的一幅图像，像素值为 0 对应的图像为黑色，像素值为 1 对应的图像为白色。由于二值图像没有中间过度的灰色，因此只能表示文字或图像轮廓。二值图像占用的空间比较少。

## 2.4.2　图像类型的转换

在数字图像处理中，经常需要转换图像类型，在 MATLAB 中有一些函数可以实现图像类型之间的转换。

### 1. RGB 图像转换为灰度图像

在 MATLAB 中，函数 rgb2gray()可以将 RGB 图像转换为灰度图像，格式如下。

```
gray =rgb2gray(A)
```

A 为输入的 RGB 图像，gray 是转换后输出的灰度图像。RGB 图像的每个像素由 3 个颜色分量值表示，灰度图像每个像素只有一个值。分别将 RGB 图像中 3 个分量的值乘以某个比例再相加，即可得到对应的灰度图像的像素值。转换后宽、高不变，数值类型不变。

rgb2gray 就是按照如下公式比例进行转换的。

```
gray1 = 0.29894 *A(:,:,1) + 0.58704 * A(:,:,2) + 0.11402 * A(:,:,3);
```

注：上述公式只是为了说明转换过程，由于精度的原因，按照上述公式转换后的结果和使用函数 rgb2gray()转换后得到的结果会有误差，建议读者以后使用函数 rgb2gray()进行转换。

下面我们通过 Code12 说明如何利用函数 rgb2gray()和公式将 RGB 图转换为灰度图像。

Code12：ch2_12.m

```
close all;clear all;clc;
rgb=imread('1.jpg');  %读取 RGB 图像
gray=rgb2gray(rgb); %使用函数将 RGB 图像转换为灰度图像
%使用转换公式转换
gray1=0.29894 * rgb(:,:,1) + 0.58704 *rgb(:,:,2) + 0.11402
*rgb(:,:,3);
figure,
subplot(1,3,1),imshow(rgb);xlabel('(a)原真彩色图像');
subplot(1,3,2),imshow(gray);xlabel('(b)转换后灰度图像 1');
subplot(1,3,3),imshow(gray1);xlabel('(c)转换后灰度图像 2');
```

运行 Code12 后，结果如图 2-19 所示。图（a）是原真彩色图像，图（b）是利用函数 rgb2gray()转换后得到的灰度图像，图（c）是利用公式转换后得到的灰度图像。





（a）原真彩色图像　　（b）转换后灰度图像 1　　（c）转换后灰度图像 2

图 2-19　Code12 的运行结果

### 2. 索引图像转换为灰度图像

在 MATLAB 中，函数 ind2gray()可以将索引图像转换为灰度图像，格式如下。

```
gray=ind2gray(X,map)
```

在索引图像转换为灰度图像的过程中，只保留索引图像的亮度，去除了颜色和饱和度。输入的索引图像数据类型无论是 double 还是 uint8，转换后的灰度图像数据类型都是 double。下面我们通过 Code13 来说明如何利用函数 ind2gray()进行索引图像到灰度图像的转换。

Code13：ch2_13.m

```
close all;clear all;clc;
[X,map]=imread('tree.tif'); %读取索引图像
gray=ind2gray(X,map);  %将索引图像转换为灰度图像
figure,
imshow(X,map);xlabel('(a)原索引图像');
figure,
imshow(gray);xlabel('(b)转换后的灰度图像');
```

运行 Code13 后，结果如图 2-20 所示。




（a）原索引图像　　（b）转换后的灰度图像

图 2-20　Code13 的运行结果

### 3. 矩阵转换为灰度图像

在 MATLAB 中，一幅图像由一个数据矩阵表示，数据矩阵中每个数值都在限定的范围内。对于任意的一个矩阵，只要把矩阵中的数值转换为限定的取值范围内的数值，就可以把一个矩阵转换为一幅灰度图像。函数 mat2gray()就可以将一个矩阵转换为一幅灰度图像，格式如下。

```
Gray=mat2gray(mat,[min,max]);
```

输入一个矩阵 mat，矩阵中的数值小于 min 会被转换为 0，即转换为黑点；大于 max 会被转换为 1，即转换为白点；其他值转换为不同程度的灰度值。min 和 max 可以选择性地进行设置，如果不设置，则默认 min 对应矩阵 mat 中的最小值，max 对应矩阵 mat 中的最大值。下面我们通过 Code14 来说明如何利用函数 mat2gray()将矩阵转换为灰度图像。

Code14：ch2_14.m

```
close all; clear all;clc;
mat=10 +1000*rand(300,300);
 %产生 300 行、300 列的矩阵，数值为随机数，取值范围是[10,1000]
gray=mat2gray(mat);
figure,
imshow(gray);xlabel('矩阵转换为灰度图像')
```

运行 Code14 后，结果如图 2-21 所示，该图是一幅灰度图像。

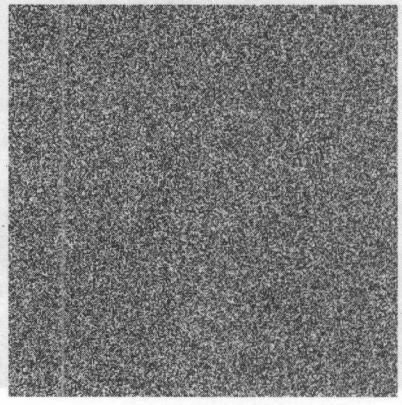


图 2-21　Code14 的运行结果

### 4. 索引图像转换为 RGB 图像

在 MATLAB 中，函数 ind2rgb()可以将索引图像转换为 RGB 图像，格式如下。

```
rgb=ind2rgb(X,map);
```

X 和 map 分别是索引图像的数据矩阵和调色板矩阵，转换过程就是将每个像素在调色板中对应的序号的颜色赋值给一个三维数组。无论输入的索引图像的数据类型是 uint8、uint16 还是 double，转换后 RGB 图像的数据类型都是 double。下面我们通过 Code15 来说明如何利用函数 ind2rgb()将索引图像转换为 RGB 图像。

Code15：ch2_15.m

```
close all;clear all;clc;
[X,map]=imread('tree.tif'); %读取索引图像
rgb=ind2rgb(X,map);  %索引图像转换为 RGB 图像
figure,
imshow(X,map);xlabel('(a)原索引图像');
figure,
imshow(rgb);xlabel('(b)转换后的 RGB 图像');
```

运行 Code15 后，结果如图 2-22 所示。图（a）是原索引图像，图（b）是通过函数 ind2rgb()转换后的 RGB 图像。



（a）原索引图像　　（b）转换后的 RGB 图像

图 2-22　Code15 的运行结果

### 5. RGB 图像转换为索引图像

上面介绍了函数 ind2rgb()可以将索引图像转换为 RGB 图像。在 MATLAB 中，也有类似的函数 rgb2ind()，可以将 RGB 图像转换为索引图像。函数 rgb2ind()有几种不同的转换方法，即均匀量化法、最小方差量化法和颜色表近似法，不同的

转换方法调用函数 rgb2ind()的方法不同，格式如下。

均匀量化法：

```
[X,map]=rgb2ind(rgb,tol)
```

rgb 是待转换的 RGB 图像，tol 决定了转换后索引图像的调色板矩阵 map 中的颜色种类，调色板矩阵 map 包含的颜色种类至少有[floor(1/tol)+1]$^3$ 种。

最小方差量化法：

```
[X,map]=rgb2ind(rgb,N)
```

N 表示生成的索引图像的调色板矩阵 map 中至少有 $N$ 中颜色。

颜色表近似法：

```
X=rgb2ind(rgb,map)
```

下面我们通过 Code16 来说明如何利用函数 rgb2ind()将 RGB 图像转换为索引图像。

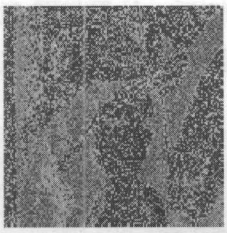
Code16：ch2_16.m

```
close all;clear all;clc;
rgb=imread('1.jpg'); %读取 RGB 图像
[x1,map1]=rgb2ind(rgb,0.3); %均匀量化法
[x2,map2]=rgb2ind(rgb,80); %最小方差量化法
map3=colorcube(100);
x3=rgb2ind(rgb,map3);  %颜色表近似法
figure,
subplot(2,2,1);imshow(rgb); xlabel('(a)原 RGB 图像')
subplot(2,2,2);imshow(x1,map1);xlabel('(b)转换后的索引图像 1')
subplot(2,2,3);imshow(x2,map2);xlabel('(c)转换后的索引图像 2')
subplot(2,2,4);imshow(x3,map3);xlabel('(d)转换后的索引图像 3')
```

运行 Code16 后，结果如图 2-23 所示。图（a）是原 RGB 图像，图（b）是均匀量化法转换后的索引图像，图(c)是最小方差量化法转换后的索引图像，图(d)是颜色表近似法转换后的索引图像。



（a）原 RGB 图像



（b）转换后的索引图像 1



（c）转换后的索引图像 2



（d）转换后的索引图像 3

图 2-23　Code16 的运行结果

### 6. 灰度图像转换为索引图像

在 MATLAB 中，函数 gray2ind()可以将灰度图像转换为索引图像，格式如下。

```
[X,map]=gray2ind(gray,n)
```

n 表示灰度级数，如果不设置，则默认为 64。

下面我们通过 Code17 来说明如何利用函数 gray2ind()将灰度图像转换为索引图像。

Code17：ch2_17.m

```
close all;clear all;clc;
gray=imread('yy.jpg'); %读取灰度图像
[x1,map1]=gray2ind(gray); %灰度图像转换为索引图像，不设置灰度级
[x2,map2]=gray2ind(gray,5);  %灰度图像转换为索引图像，灰度级为 5
figure,
imshow(gray);xlabel('(a)原灰度图像');
figure,
imshow(x1,map1);xlabel('(b)转换后的索引图像 1');
figure,
imshow(x2,map2);xlabel('(c)转换后的索引图像 2');
```

运行 Code17 后，结果如图 2-24 所示。图（a）是原灰度图像，图（b）是默认灰度级转换后的索引图像，图（c）是灰度级为 5 转换后的索引图像。



（a）原灰度图像　　（b）转换后的索引图像 1　　（c）转换后的索引图像 2

图 2-24　Code17 的运行结果

**7．各种类型的图像转换为二值图像**

二值图像中数据的值都是 1 或 0，将其他类型的图像矩阵转换为二值图像的主要工作是：确定哪些值转换为 1，哪些值转换为 0。通常我们设置一个阈值 level，如果小于 level 的值转化为 0，则其他都转换为 1，这样就可以将其他类型的图像转换为对应的二值图像。在 MATLAB 中，函数 im2bw()可以将其他类型的图像转换为二值图像，但调用格式不同，具体如下。

灰度图像和 RGB 图像转换为二值图像，格式为：

```
BW=im2bw(img,level)
```

level 的取值范围为(0,1)。若 level=0.5，则所有小于 255×0.5=127.5 的值转换为 0，其他值都转换为 1。

如果 img 是 RGB 图像，调用函数 im2bw()，RGB 图像首先会被直接转换为灰度图像，然后根据 level 值相应地转换为 0 或 1。

索引图像转换为二值图像，格式为：

```
BW=im2bw(X,map,level)
```

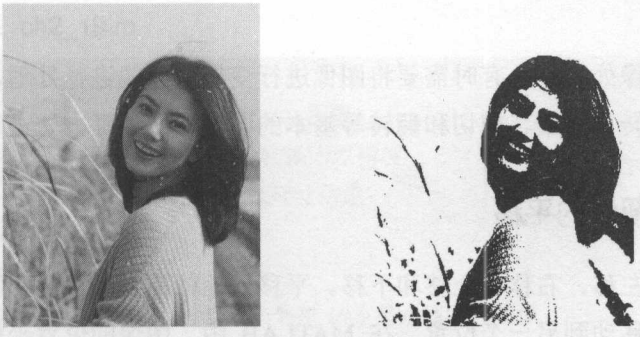
Level 的取值范围为[0,1]。如果输入的是索引图像的数据矩阵 X 和调色板矩

阵 map，那么首先它们会被转换为灰度图像，然后根据 level 值相应地转换为 0 或 1。下面我们通过 Code18 来说明如何利用函数 im2bw()将各种类型的图像转换为二值图像。

Code18：ch2_18.m

```
close all;clear all;clc;
gray=imread('yy.jpg'); %读取灰度图像
rgb=imread('1.jpg');  %读取 RGB 图像
[X,map]=imread('tree.tif');  %读取索引图像
gray_BW=im2bw(gray);  %灰度图像转换为二值图像
rgb_BW=im2bw(rgb);   %RGB 图像转换为二值图像
index_BW=im2bw(X,map);  %索引图像转换为二值图像
figure,
subplot(1,2,1),imshow(gray);xlabel('(a)原灰度图像');
subplot(1,2,2),imshow(gray_BW);xlabel('(b)转换后的二值图像');
figure,
subplot(1,2,1),imshow(rgb);xlabel('(a)原 RGB 图像');
subplot(1,2,2),imshow(rgb_BW);xlabel('(b)转换后的二值图像');
figure,
subplot(1,2,1),imshow(X,map);xlabel('(a)原索引图像');
subplot(1,2,2),imshow(index_BW);xlabel('(b)转换后的二值图像');
```

运行 Code18 后，结果如图 2-25 所示，有三组图，分别为灰度图像、RGB 图像和索引图像到二值图像的转换。



（a）原灰度图像　　（b）转换后的二值图像

Figure1：原灰度图像和转换后的二值图像的比较

图 2-25　Code18 的运行结果



（a）原 RGB 图像



（b）转换后的二值图像

Figure2：原 RGB 图像和转换后的二值图像的比较



（a）原索引图像



（b）转换后的二值图像

Figure3：原索引图像和转换后的二值图像的比较

图 2-25　Code18 的运行结果（续）

## 2.5　图像变换处理

在数字图像处理中，有时需要将图像进行变换后才能进行处理，本节主要对图像平移、旋转、缩放、剪切和翻转等基本的图像变换处理方法进行介绍。

### 2.5.1　图像的平移

平移包括左移、右移、上移和下移。平移处理就是根据需要，将图像中每个位置的像素值移动到另一个位置。在 MATLAB 中，图像的位置坐标表示法和数学上的不太一致，在 MATLAB 中 $x$ 轴是从左到右，$y$ 轴是从上到下，原点在左上角。图 2-26 中的红点在 MATLAB 中的坐标为(100,50)，即距离图像左边界的距离

为 100 像素，距离图像上边界的距离为 50 像素。

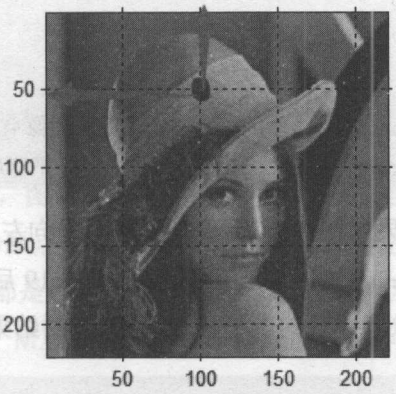



图 2-26　图像在 MATLAB 中的坐标形式

图像平移就是平移图像中的像素点的位置，首先需要创建一个新图像 newimg 存放平移后的图像，newimg 和原图像的尺寸信息一致。通常，我们设 newimg=ones(size(img))或者 newimg=zeros(size(img))，这样便可保证平移后空白的部分为纯白色或纯黑色。

假设 *a* 表示上下平移量，*b* 表示左右平移量。

如果对一个灰度图像进行平移，则 newimg(i,j)=img(i+a,j+b)。

如果对一个真彩色图像进行平移，则 newimg(i,j,:)=img(i+a,j+b,:)。

*a* 大于 0 表示上移，*a* 小于 0 表示下移。*b* 大于 0 表示左移，*b* 小于 0 表示右移。下面我们通过 Code19 来说明如何对图像进行平移处理。

Code19：ch2_19.m

```
close all;clear all;clc;
img=imread('1.jpg'); %读取 RGB 图像
img=im2double(img); %将图像转换为双精度
[m,n,d]=size(img); %获得图像尺寸信息
a=0;  %垂直平移量
b=50; %水平平移量
newimg=ones(m,n,d);  %创建一个和 img 尺寸相同的图像
for i=1:m
    for j=1:n
        if (((i+a)>=1)&&((i+a)<=m)&&((j+b)>=1)&&((j+b)<=n))
```

```
            newimg(i,j,:)=img(i+a,j+b,:);
        end
    end
end
figure,
subplot(1,2,1),imshow(img);xlabel('(a)原图像')
subplot(1,2,2),imshow(newimg);xlabel('(b)平移后的图像')
```

运行 Code19 后，结果如图 2-27 所示，图（b）是向左平移 50 像素后的结果。修改 Code19 中的平移量：$a$=50，$b$=−50，再运行 Code19 后，结果如图 2-28 所示，图（b）是向右平移 50 像素并且向上平移 50 像素后的结果。



（a）原图像　　（b）平移后的图像

图 2-27　图像向左平移 50 像素



（a）原图像　　（b）平移后的图像

图 2-28　图像向右平移 50 像素并向上平移 50 像素

读者可根据需求自行修改垂直平移量 $a$ 的值和水平平移量 $b$ 的值。

Code19 在对图像平移的同时，对超出边界范围的图像进行了截取，如果不想截取，则可根据需要的平移量在水平或垂直方向上扩展 newimg 的尺寸。本书在此不做更多介绍。

## 2.5.2　图像的旋转

在数字图像处理中，图像旋转是一种不可或缺的图像变换操作之一。图像旋转就是以图像中心作为原点，将图像按照顺时针或者逆时针旋转一定角度。由于在 MATLAB 中图像都是以数值矩阵的形式存储的，因此，图像的旋转本质上就是像素点的旋转。在 MATLAB 中，函数 imrotate()可以实现图像的旋转，格式如下。

```
Newimg=imrotate(img,angle,method,bbox)
```

Newimg 是旋转后的图片，img 是旋转前的图片。

angle 表示旋转的角度，angle>0，图像逆时针旋转；angle<0，图像顺时针旋转。

method 表示旋转时采用的方法，如果不设置，则默认 method=nearest，表示最近邻插值。也可以选择其他方法，如 method=bilinear，表示双线性插值；method=bicubic，表示双三次插值。

bbox 表示返回图像的大小，bbox=crop 表示输出图像与原图像尺寸一致；bbox=loose 表示输出图像有足够的空间包含旋转后的图像。若不设置 bbox 的值，则默认 bbox=loose。

下面我们通过 Code20 来说明如何利用函数 imrotate()实现图像的旋转。

Code20：ch2_20.m

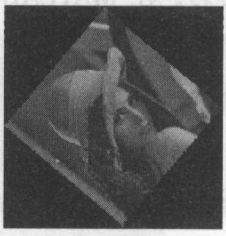
```
close all;clear all;clc;
img=imread('1.jpg');  %读取图片
newimg1=imrotate(img,50);  %逆时针旋转 50°
newimg2=imrotate(img,-50,'bilinear');  %顺时针旋转 50°,旋转时使用的方法为 bilinear
newimg3=imrotate(img,50,'crop'); %逆时针旋转 50°，限制输出图像和原图像尺寸一致
figure,
subplot(221);imshow(img); xlabel('(a)原图像');
```

```
subplot(222);imshow(newimg1);xlabel('(b)旋转后的图像 1');
subplot(223);imshow(newimg2);xlabel('(c)旋转后的图像 2');
subplot(224);imshow(newimg3);xlabel('(d)旋转后的图像 3');
```

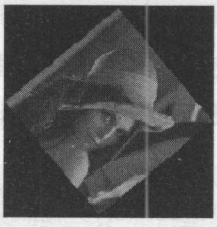
运行 Code20 后，结果如图 2-29 所示，可以看到原图像和各种不同旋转处理后的效果。图（a）是原图像；图（b）是 angle=50，即逆时针旋转 50°后的图像；图（c）是 angle=−50，method=bilinear，即顺时针旋转 50°并且旋转方法为双线性插值；图（d）是 angle=50，bbox=crop，即逆时针旋转 50°，并且旋转后的图像和原图像尺寸一致，越界的部分图像进行截取处理。图（b）和图（c）中转换后的图像是完整的。



（a）原图像



（b）旋转后的图像 1



（c）旋转后的图像 2



（d）旋转后的图像 3

图 2-29　Code20 的运行结果

## 2.5.3　图像的缩放

图像缩放是将原图像按照一定的比例进行缩小或放大。如果垂直方向和水平方向的缩放比例是一样的，则图像是全比例缩放，不会变形；若不一致，则缩放后的图像会发生几何畸形。在 MATLAB 中，函数 imresize()可以实现图像的缩放，格式如下。

```
Newimg=imresize(img,m);
```

img 是原图像，可以是灰度图像、RGB 图像或者二值图像。Newimg 是缩放后的图像。m 是缩放的比例，当 0<m<1 时，图像缩小；当 m>1 时，图像放大。

```
Newimg=imresize(img,[rows,clos])
```

img 是原图像，可以是灰度图像、RGB 图像或者二值图像。Newimg 是缩放后的图像。rows 和 clos 分别是缩放后的图像 Newimg 的行和列。

```
[newX newmap]=imresize(X,map,m)
```

和

```
[newX newmap]=imresize(X,map, [rows,clos])
```

这两种格式适用于索引图像的缩放，m 和 rows、clos 的意义同上。

下面我们通过 Code21 来说明如何利用函数 imresize()实现图像的缩放。

Code21：ch2_21.m

```
close all;clear all;clc;
img=imread('1.jpg');  %读取 RGB 图像 img
[X,map]=imread('tree.tif'); %读取索引图像
newimg1=imresize(img,0.2);  %将 img 图像缩小
newimg2=imresize(img,2);  %将 img 图像放大
newimg3=imresize(img,[200,100]);  %限制输出图像的尺寸，进行缩放
[newX,newmap]=imresize(X,map,0.5);  %索引图像进行缩小
figure,
subplot(2,2,1);imshow(img);xlabel('(a)原 RGB 图像');
subplot(2,2,2);imshow(newimg1); xlabel('(b) RGB 图像缩放变换图像 1');
subplot(2,2,3);imshow(newimg2); xlabel('(c)RGB 图像缩放变换图像 2');
subplot(2,2,4);imshow(newimg3); xlabel('(d)RGB 图像缩放变换图像 3');
figure;imshow(X,map); xlabel('(a)原索引图像');
figure;imshow(newX,newmap); xlabel('(b)缩小后的索引图像');
```

运行 Code21 后，结果如图 2-30 所示。结果分为两组图，Figure1 是对 RGB 图像的缩放，图（a）是原 RGB 图像，图（b）是缩小后的 RGB 图像，图（c）是放大后的 RGB 图像，图（d）是限制图像输出尺寸后的 RGB 图像。Figure2 是对索引图像的缩放，图（a）是原索引图像，图（b）是缩小后的索引图像。



（a）原 RGB 图像



（b）RGB 缩放变换图像 1



（c）RGB 缩放变换图像 2



（d）RGB 缩放变换图像 3

Figure1：RGB 图像的缩放



（a）原索引图像



（b）缩小后的索引图像

Figure 2：索引图像的缩放

图 2-30　Code21 的运行结果

函数 imresize()还有一些可以设置的参数，如缩放方法 method 和缩放特性 parameter，关于这些参数的使用，读者可以查阅 MATLAB 的帮助文档进行更多的学习。

### 2.5.4　图像的剪切

在数字图像处理中，我们通常需要获取一幅图像中我们感兴趣的部分。如果我们需要获取图 2-31 中包含人脸的那一部分，即方框区域内的图像，那么就需要对图像进行剪切处理。在 MATLAB 中，函数 imcrop()可以实现图像的剪切，格式如下。

```
newimg=imcrop(img,rect);
```

newimg 是剪切后的图像。img 是原图像，如果原图像是灰度图像、RGB 图像或者二值图像，那么 img 就是图像矩阵；如果原图像是索引图像，则 img=X，即[X,map]=imread('索引图像')中的 X 矩阵。rect 是剪切的区域，其形式为[$x$,$y$,$w$,$h$]，($x$,$y$)是剪切区域的左上角坐标，$w$ 是剪切区域的宽度，$h$ 是剪切区域的高度。

```
[newimg,rect]=imcrop(img)
```

和

```
[X,Y,newimg,rect]=imcrop(img)
```

img 为原图像，如果是索引图像，则 img 就是数据矩阵 X。

使用这两种格式，首先会显示 img，然后会出现剪切工具，利用鼠标进行剪切后，会把剪切区域信息返回给 newimg、rect 和 X、Y。



图 2-31　对方框区域内的图像进行剪切

下面我们通过 Code22 来说明在给定剪切区域范围的情况下，如何利用函数 imcrop()实现图像的剪切。

Code22：ch2_22.m

```
close all;clear all;clc;
[X,map]=imread('tree.tif');  %读取索引图像
rect=[100,100,100,100];  %截取的区域
newimg1=imcrop(X,rect);  %在索引图像上截取 rect 区域
img=imread('1.jpg');  %读取 RGB 图像
newimg2=imcrop(img,rect);  %在 RGB 图像上截取 rect 区域
figure,
imshow(X,map);xlabel('(a)原索引图像');
figure,
imshow(newimg1);xlabel('(b)截取索引图像的部分图像');
figure,
subplot(1,2,1);imshow(img);xlabel('(a)原 RGB 图像');
subplot(1,2,2);imshow(newimg2);xlabel('(b)截取 RGB 图像的部分图像');
```

运行 Code22 后，结果如图 2-32 所示。程序首先读入索引图像，设置剪切区域为[100,100,100,100]，使用函数 imcrop()在索引图像上截取该区域。然后读入 RGB 图像，使用函数 imcrop()在 RGB 图像上截取相同的区域。最后在 Figure1 中显示原索引图像和对应的截取区域，在 Figure2 中显示原 RGB 图像和对应的截取区域。



（a）原索引图像　　（b）截取索引图像的部分图像

Figure1：原索引图像和截取的部分图像

图 2-32　Code22 的运行结果



（a）原 RGB 图像



（b）截取 RGB 图像的部分图像

Figure2：原 RGB 图像和截取的部分图像

图 2-32　Code22 的运行结果（续）

下面我们通过 Code23 来说明在没有给定剪切区域范围的情况下，如何把函数 imcrop()当作剪切工具进行图像的剪切。

**Code23：ch2_23.m**

```
close all;clear all;clc;
img=imread('yy.jpg');
[newimg,rect]=imcrop(img); %会调用剪切工具进行剪切
figure,
subplot(1,2,1);imshow(img);xlabel('(a)原 RGB 图像');
hr = rectangle('Position', rect);
set(hr, 'EdgeColor', [1 0 0]);
set(hr, 'LineWidth', 3)
subplot(1,2,2);imshow(newimg);xlabel('(b)截取 RGB 图像的部分图像');
```

运行 Code23 后，首先会显示原图像，并且出现剪切工具，可通过滑动鼠标选择要截取的图像区域，如图 2-33 所示。选择好要截取的区域后，双击该区域，此时截图完成，可以看到工作空间增加了 rect 和 newimg。rect 是截取图像的范围，如图 2-34 所示；newimg 是截取的图像矩阵。最后显示原图像和截图部分的图像，并且以截取范围在原图像上作一个矩形框，如图 2-35 所示。

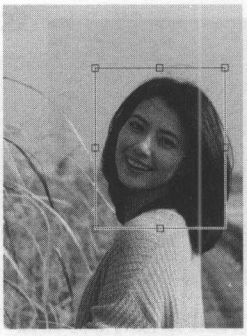
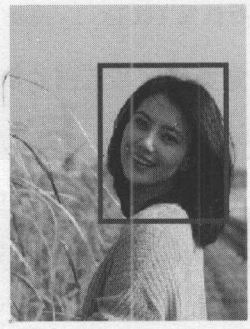


图 2-33　滑动鼠标选择要截取的图像区域

rect <1x4 double>

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 115.5100 | 78.5100 | 160.9800 | 196.9800 |
| 2 | | | | |
| 3 | | | | |

如图 2-34　截取图像的范围



（a）原 RGB 图像



（b）截取 RGB 图像的部分图像

图 2-35　Code23 的运行结果

## 2.5.5　图像的翻转

在数字图像处理中，图像的镜像就是对图像进行翻转，分为水平镜像和垂直镜像。水平镜像是以图像的垂直中心轴，将图像左右部分进行翻转；垂直镜像是以图像的水平中心轴，将图像上下部分进行翻转。在 MATLAB 中，函数 fliplr()可以实现矩阵的水平翻转，函数 flipud()可以实现矩阵的垂直翻转。由于在 MATLAB 中，图像是以矩阵的形式存在的，因此可使用函数 fliplr()和 flipud()进行图像的翻转，格式如下。

```
Newimg=fliplr (img);
Newimg=flipud(img);
```

如果是索引图像，则只翻转它的数据矩阵 X 即可（[X,map]=imread(ind)）。

下面通过 Code24 来说明如何利用函数 fliplr()和 flipud()进行图像的翻转。

Code24：ch2_24.m

```
close all;clear all;clc;
img=imread('yy.jpg'); %读取灰度图像
newimg1=fliplr(img); %对灰度图像进行水平翻转
newimg2=flipud(img);%对灰度图像进行垂直翻转
newimg5=fliplr(newimg2); %对灰度图像进行水平垂直翻转
[X,map]=imread('tree.tif'); %读取索引图像
newimg3=fliplr(X);  %对索引矩阵的数据矩阵 X 进行水平翻转
newimg4=flipud(X);%对索引矩阵的数据矩阵 X 进行垂直翻转
newimg6=fliplr(newimg4);%对索引矩阵的数据矩阵 X 进行水平垂直翻转
figure,
subplot(2,2,1);imshow(img);xlabel('(a)原灰度图像');
subplot(2,2,2);imshow(newimg1,map);xlabel('(b)水平翻转后的图像')
subplot(2,2,3);imshow(newimg2);xlabel('(c)垂直翻转后的图像')
subplot(2,2,4);imshow(newimg5);xlabel('(d)水平垂直翻转后的图像')
figure,
subplot(2,2,1);imshow(X,map);xlabel('(a)原索引图像')
subplot(2,2,2);imshow(newimg3,map);xlabel('(b)水平翻转后的图像')
subplot(2,2,3);imshow(newimg4,map);xlabel('(c)垂直翻转后的图像')
subplot(2,2,4);imshow(newimg6,map);xlabel('(d)水平垂直翻转后的图像')
```

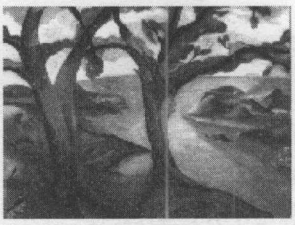
运行 Code24 后，结果如图 2-36 所示，Figure1 是对灰度图像的各种翻转，Figure2 是对索引图像的各种翻转。



（a）原灰度图像　（b）水平翻转后的图像　（c）垂直翻转后的图像　（d）水平垂直翻转后的图像

Figure1：灰度图像的各种翻转

图 2-36　Code24 的运行结果



（a）原索引图像



（b）水平翻转后的图像



（a）垂直翻转后的图像



（b）水平垂直翻转后的图像

Figure2：索引图像的各种翻转

图 2-36　Code24 的运行结果（续）

## 2.6　图像的噪声和滤波

在数字图像处理中，经常会遇到一些模糊、失真或者有干扰的图像，这些造成图像不清晰的因素就是图像的噪声。图像的噪声可分为加性噪声、乘性噪声和量化噪声，常见的噪声模型有高斯噪声、椒盐噪声、均匀分布噪声、指数分布噪声和 Gamma 分布噪声。为了获得清晰的、高质量的图像，需要对有噪声的图像进行去噪，比较经典的去噪算法有均值滤波、自适应维纳滤波、中值滤波、形态学噪声滤波和小波滤波。本节主要对常见的噪声模型和经典的去噪算法进行介绍。

### 2.6.1　常见的噪声模型

#### 1．高斯噪声

高斯噪声是一种源于电子电路噪声和由低照明度或高温带来的传感器噪声。高斯噪声是一种比较常见的噪声，它的概率密度呈正态分布。

在 MATLAB 中，函数 imnoise()可以对图像添加高斯噪声，有三种方式，格式如下。

第一种：

```
newimg=imnoise(img,' gaussian',m,v);
```

newimg 是添加高斯噪声后的图像，img 是原图像，gaussian 表示噪声类型为高斯噪声，*m* 为噪声的均值，*v* 是噪声的方差。*m* 和 *v* 如果不设置，则使用系统默认参数。

第二种：

```
newimg=imnoise(img,'localvar', V);
```

newimg 是添加高斯噪声后的图像，img 是原图像，*V* 是一个和 img 相同尺寸的数组，*V* 中的每个元素表示在 img 对应位置添加的高斯噪声的方差，此时高斯噪声的均值为 0。

第三种：

```
newimg=imnoise(img,'localvar', h,v);
```

newimg 是添加高斯噪声后的图像，img 是原图像，***h*** 是一个表示图像亮度的向量，亮度值在[0,1]。*v* 和 ***h*** 的长度相同，***h*** 的每个值表示和 *v* 对应位置的亮度上添加的高斯噪声的方差。

下面我们通过 Code25 来说明如何在图像上添加高斯噪声。

Code25：ch2_25.m

```
close all;clear all;clc;
img=imread('1.jpg');  %读取图片
img=im2double(img);
V=zeros(size(img)); %创建一个和 img 相同尺寸的数组 V
V(1:50,:)=0.05; %数组 V 的前 50 行元素值设置为 0.05
newimg1=imnoise(img,'gaussian',0,0.01);  %添加方差为 0.01 的噪声
newimg2=imnoise(img,'localvar',V); %根据不同位置添加高斯噪声
 %根据亮度添加高斯噪声
newimg3=imnoise(img,'localvar',[0.1,0.2,0.3],[0.01,0.02,0.03]);

%显示图片
```

```
figure,
subplot(2,2,1);imshow(img); xlabel('(a)原图像');
subplot(2,2,2);imshow(newimg1);xlabel('(b)添加高斯噪声后的图像 1');
subplot(2,2,3);imshow(newimg2);xlabel('(c)添加高斯噪声后的图像 2');
subplot(2,2,4);imshow(newimg3);xlabel('(d)添加高斯噪声后的图像 3');
```

运行 Code25 后，结果如图 2-37 所示。该程序使用三种不同的方式添加高斯噪声，图（a）是原图像，图（b）是根据噪声的均值和方差添加高斯噪声后的图像，图（c）是根据像素位置添加高斯噪声后的图像，图（d）是根据图像的亮度添加高斯噪声后的图像。



（a）原图像



（b）添加高斯噪声后的图像 1



（c）添加高斯噪声后的图像 2



（d）添加高斯噪声后的图像 3

图 2-37　Code25 运行后的结果

### 2. 椒盐噪声

椒盐噪声是一种只有两种灰度值作为噪声出现在图像上的噪声，因此也称为双极脉冲噪声。负脉冲时黑点出现，正脉冲时盐点（白点）出现。

去除椒盐噪声最常用的算法是中值滤波。

对图像 img 添加噪声，类型可以为：gaussian（高斯噪声）、salt&pepper（椒盐噪声）、speckle（乘性噪声）、poisson（泊松噪声）。param 是对应类型噪声的参数，如果不设置，则使用系统默认的参数。

在 MATLAB 中，也可以使用函数 imnoise()产生椒盐噪声，格式如下。

```
newimg=imnoise(img, 'salt & pepper',d)
```

newimg 是添加椒盐噪声后的图像；img 是原图像；salt & pepper 表示噪声的类型为椒盐噪声；*d* 为椒盐噪声的密度，表示椒盐噪声占图像像素总分的百分比。

下面我们通过 Code26 来说明如何利用函数 imnoise()产生椒盐噪声。

Code26：ch2_26.m

```
close all;clear all;clc;
img=imread('yy.jpg'); %读取图像
img=im2double(img);  %将图像转换为双精度浮点型
newimg1=imnoise(img,'salt & pepper',0.01); %添加密度为 0.01 的椒盐噪声
newimg2=imnoise(img,'salt & pepper',0.08); %添加密度为 0.08 的椒盐噪声

figure,
subplot(1,2,1);imshow(newimg1);xlabel('(a)添加椒盐噪声后的图像 1')
subplot(1,2,2);imshow(newimg2);xlabel('(b)添加椒盐噪声后的图像 2')
```

运行 Code26 后，结果如图 2-38 所示。图（a）的噪声密度为 0.01，图（b）的噪声密度为 0.08。从中可以看出，噪声密度越大，噪声越多。黑点代表胡椒点，白点代表盐点。



（a）添加椒盐噪声后的图像 1　（b）添加椒盐噪声后的图像 2

图 2-38　Code26 的运行结果

此外，函数 imnoise()还可以产生其他类型的噪声，如乘性噪声、泊松噪声等，本书不多做介绍，读者可查阅 MATLAB 帮助文档进行学习。

## 2.6.2 经典的去噪算法

### 1. 均值滤波

均值滤波是一种简单的去噪方法，对于图像中的坐标为$(x,y)$的像素，以该点为中心创建一个矩形框，计算该矩形框中像素的平均值，用得到的平均值替换像素点$(x,y)$的值。在 MATLAB 中，可以通过函数 fspecial('average')得到均值滤波算子，然后利用函数 imfilter()对含噪声的图像进行滤波。

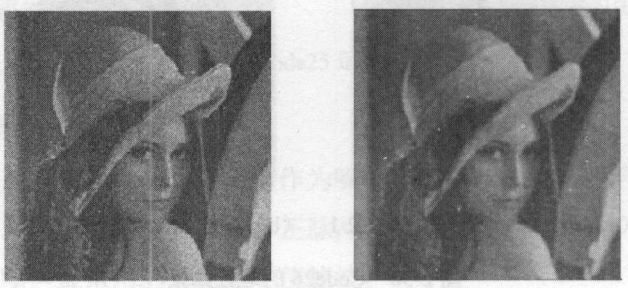
下面我们通过程序 Code27 来说明如何利用函数 fspecial()和函数 imfilter()对包含噪声的图像进行均值滤波。

Code27:ch2_27.m

```
close all;clear all;clc;
img=imread('1.jpg'); %读取图像
img=im2double(img); %转换为双精度浮点型
img=imnoise(img,'gaussian',0.02); %添加高斯噪声
psf=fspecial('average',3); %产生 psf 算子
newimg=imfilter(img,psf); %进行均值滤波

figure ,
subplot('1,2,1');imshow(img);xlabel('(a)含高斯噪声的图像');
subplot('1,2,2');imshow(newimg);xlabel('(b)均值滤波后的图像');
```

运行 Code27 后，结果如图 2-39 所示。可以看到，进行均值滤波后的图像变得模糊，这是因为均值滤波破坏了图像的细节部分，这也是均值滤波的一个缺点。



（a）含高斯噪声的图像　　（b）均值滤波后的图像

图 2-39　Code27 的运行结果

### 2. 自适应维纳滤波

在 MATLAB 中，函数 wiener2()可以对含噪声的图片进行自适应维纳滤波，还可以估计噪声。该函数由图像的局部方差决定滤波器的输出结果，使用格式如下。

```
newimg=wiener2(img,[m,n],noise);
```

img 是包含噪声的图像；[m,n]是采用的窗口大小，默认为[3,3]；noise 是噪声的能量；newimg 是滤波后的图像。

```
[newimg,noise]=wiener2(img,[m,n])
```

该函数用来估计图像中的噪声。

下面我们通过 Code28 来说明如何对包含噪声的图像进行自适应维纳滤波。

Code28：ch2_28.m

```
close all;clear all;clc;
img=imread('1.jpg');  %读取一幅 RGB 图像
img=rgb2gray(img);  %转换为灰度图像
img=imnoise(img,'gaussian',0,0.01);  %在图像上添加噪声
[newimg,noise]=wiener2(img,[4,4]); %对含噪声的图像进行自适应维纳滤波

figure,
subplot('1,2,1');imshow(img);xlabel('(a)含噪声的图像');
subplot('1,2,2');imshow(newimg);xlabel('(b)自适应维纳滤波后的图像');
```

运行 Code28 后，结果如图 2-40 所示。在工作空间中，查看 noise 的值，即估计的噪声能量，noise=0.0124。



（a）含噪声的图像　　（b）自适应维纳滤波后的图像

图 2-40　Code28 的运行结果

### 3. 中值滤波

中值滤波是顺序统计滤波中的一种，顺序统计滤波中还有最小值滤波、最大值滤波。中值滤波适合去除椒盐噪声，因为它可以保留图像的边缘。对于图像上坐标为$(x,y)$的像素，以该像素为中心的一个 $m\times n$ 像素的窗口，找出该区域内中间的像素值，用于替换点$(x,y)$的像素值。在 MATLAB 中，函数 medfilt2()可实现图像的二维中值滤波，格式如下。

```
newimg=medfilt2(img,[m,n]);
```

img 是含噪声的图像；newimg 是滤波后的图像；[m,n]是窗口的大小，默认为[3,3]。

下面我们通过 Code29 来说明如何利用函数 medfilt2()对含噪声的图像进行滤波。

Code29：ch2_29.m

```
close all;clear all;clc;
img=imread('1.jpg'); %读取 RGB 图像
img=im2double(img);
img=rgb2gray(img);  %将 RGB 图像转换为灰度图像
img=imnoise(img,'salt & pepper',0.01); %添加椒盐噪声
newimg=medfilt2(img,[4,4]);  %使用中值滤波

figure,
subplot('1,2,1'); imshow(img); xlabel('(a)含椒盐噪声的图像');
subplot('1,2,2'); imshow(newimg);xlabel('(b)中值滤波后的图像');
```

运行 Code29，结果如图 2-41 所示。由图 2-41（a）和（b）可以看出，中值滤波对于含椒盐噪声的图像去噪效果较好。



（a）含椒盐噪声的图像



（b）中值滤波后的图像

图 2-41　Code29 的运行结果

# 第 3 章

# 人脸检测实战

人脸检测是计算机视觉方面的一个重要方向，是目标检测的一个特例。我们通过眼睛可以快速准确地辨别出一张图片中是否包含人脸，以及人脸的所在位置。随着计算机技术的迅速发展，人们希望可以借助计算机自动检测人脸，并把这一技术嵌入到各种应用中。

经过多年的研究，已经出现了一些较为准确的人脸检测算法。本章主要介绍 3 种人脸检测算法，分别是 DPM、LAEO、Viola&Jones。下面我们将从使用方法、算法原理和检测结果这三个方面对每种算法进行介绍，让读者能够对人脸检测技术有一个全面的了解。

## 3.1 DPM 人脸检测算法

该人脸检测算法使用的检测模型为 dpm-baseline[1]，本书称之为 DPM 人脸检测算法。它可以检测多角度的人脸，是 M.Mathias 等人在 2014 年提出的[2]。和一些比较成熟的算法相比，DPM 人脸检测算法的检测效果和它们相当，甚至可以取得更好的结果，主要原因在于：其选择了合适的训练数据，以及发现了非极大值抑

制的重要性。M.Mathias 等人公开了训练好的人脸检测模型——dpm-baseline，并提供了开源的代码。本书就是使用他们提供的开源模型和源代码进行人脸检测的。

### 3.1.1 DPM 人脸检测算法的使用

本书实现的 DPM 人脸检测算法是基于 MATLAB 语言实现的，对应的项目名称为 dpmMATLAB。该项目内包含了 DPM 算法运行所需的全部文件，分为两部分，即 dpm_face_detector 和 voc-dpm-master[3]。dpm_face_detector 中存放了人脸检测模型[1]、主要程序[4]和测试图片；voc-dpm-master 中存放了程序运行中需要调用的函数。

在运行程序之前，还需要一些配置，具体如下：

（1）首先打开 MATLAB，切换到主函数 detect_faces.m 所在的目录。

（2）单击 home→Set Path→Add to Path with Subfolders，选择 voc-dpm-master 文件夹，如图 3-1 所示，最后单击“保存”按钮并关闭。

（3）本书程序是在 Windows 64 位操作系统和 MATLAB 2012b 下运行的。如果读者的运行环境和本实验一致，则配置完成。如果读者使用的是 32 位操作系统或者在文献[3]中下载 voc-dpm-master，则使用 dpmMATLAB\change 文件夹下的.cc 文件，替换掉 voc-dpm-master 中对应的文件，并使用“mex - O xx.cc”对这类.cc 文件进行编译。编译成功后，则配置完成。

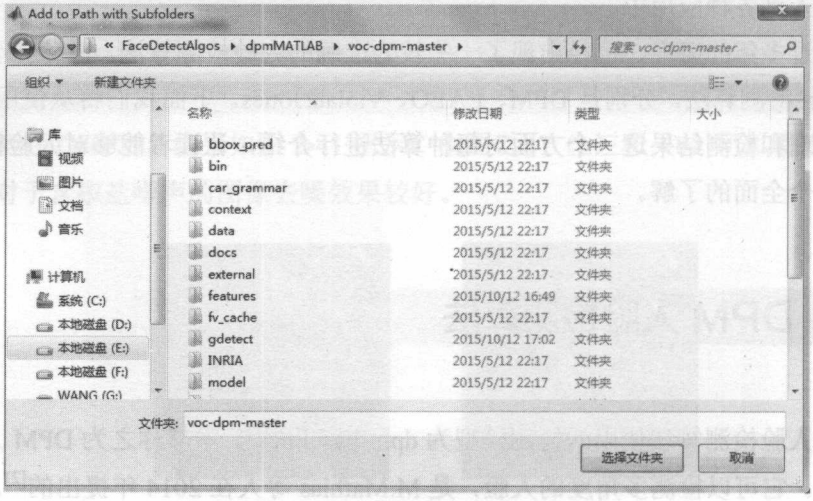



图 3-1 Add to Path with Subfolders 对话框

待检测图片在 data\original_image 文件夹下。

检测结果在 data\face_detection_results 文件夹下。

读者可根据实际情况，在 detect_faces.m 文件中，修改存放检测图片和检测结果的路径，如下：

```
images_folder_path = 'data/original_image';
results_folder_path = 'data/face_detection_results';
```

选择好路径后，运行主函数 detect_faces.m，即可使用 DPM 人脸检测算法进行人脸检测。

### 3.1.2　DPM 人脸检测算法的原理

该算法使用可变形部件模型（Deformable Part Model）目标检测框架[5]，包含一个训练好的、可以检测多角度人脸的模型 dpm-baseline.mat，对输入的检测图片在检测模型上进行处理后，得到若干个检测框，初步得到的检测框经过 NMS 处理，即可得到最终人脸检测结果。作者在文中提到，该算法检测质量好的主要原因是选择了合适的训练数据和发现了非极大值抑制的重要性[2]。

#### 1．dpm-baseline 模型的训练数据

该模型从 AFLW 数据集收集了 15 106 个样本数据进行训练，分为三类：6752 个正面脸样本（脸部上下偏移±20°），5810 个侧面脸样本（脸部左右偏移 20°～60°），2544 个侧面脸样本（脸部左右偏移 60°～100°），分别训练成 3 个组件（Component）。通过翻转，3 个组件变为 6 个组件，每个组件有一个根模板和 8 个部件，如图 3-2 所示，该图来源于文献[2]。除初始化组件外，所有训练参数都使用默认值。

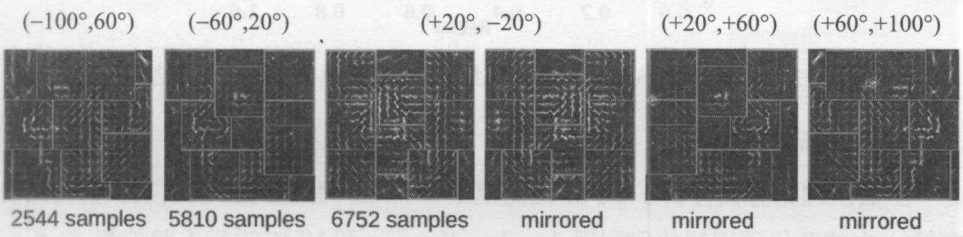



图 3-2　dpm-baseline 模型的 6 个组件

选择的样本质量和数量合适，是该算法检测效果好的一个重要因素。在检测质量方面，DPM 算法选择了多角度的人脸样本进行训练，训练好的模型可以检测多种角度的人脸。作者从精度和召回率方面评估了训练样本数量不同对检测质量的影响。结果显示，当有 500 个训练样本时，DPM 的平均精度可以达到 95%，增加训练样本个数可以在保持精度不变的情况下提高召回率。

### 2. 非极大值抑制的重要性

非极大值抑制（Non-Maximum Suppression，NMS）的目的是保证每个检测实例只有一个检测框。对于同一个检测实例，选择得分最高的检测框的同时，自动移除得分低的检测框。假设我们设置重叠阈值（Overlap Thresh）为 0.3，已经找到了一个得分高的检测框 A，比较检测框 B 和检测框 A，如果检测框 A、B 的重叠面积比总和面积（IoU）小于 0.3，则认为检测框 A、B 不是同一个检测实例，保留检测框 B，否则删除检测框 B。NMS 的重叠阈值是一个影响检测质量的重要参数。作者在论文中指出，使用不同的重叠阈值在 Pascal Faces 数据集上比较 DPM 的检测质量，发现阈值为 0.3 比默认值 0.5 的检测效果好，如图 3-3 所示，该图引自文献[2]。

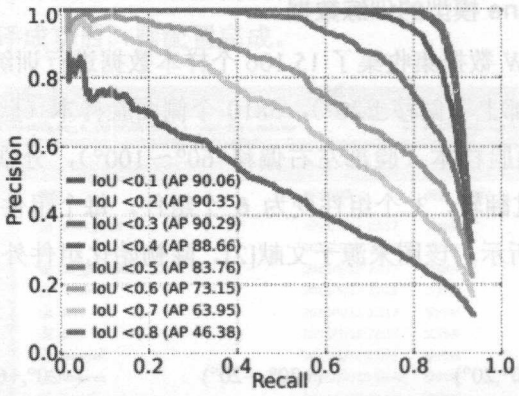



图 3-3　DPM 在 Pascal Faces 数据集上使用不同重叠阈值的检测结果

### 3. 源代码分析

上面从理论方面分析了 DPM 检测模型和 NMS 的重要性，接下来我们将结合文献[4]中的源代码详解如何利用 dpm-baseline.mat 和 NMS 进行人脸检测。Code1、Code2 中的程序均来自文献[4]。

Code1：主函数 detect_faces.m

```
images_folder_path = 'data/original_image'; %检测图片路径
results_folder_path = 'data/detect_False_result';  %检测结果路径
model_path = 'dpm_baseline.mat';%模型路径
face_model = load(model_path); %加载模型
detection_threshold = 0; %值越小，可以产生越多的检测框
nms_threshold = 0.3;  %NMS 最佳值
image_names = dir(fullfile(images_folder_path, '*.jpg'));
for i=1:numel(image_names)  %对路径下的每一张图片进行遍历
    image_name = image_names(i).name;
    image_path = fullfile(images_folder_path, image_name);%获得图
片路径
    image = imread(image_path);%读取图片
    %通过 process_face 函数，得到的检测框存放在 ds 矩阵中
    [ds, bs] = process_face(image, face_model.model,
detection_threshold, nms_threshold);
    result_path = fullfile(results_folder_path, [image_name,
'.result.png']); %检测后存放的路径和检测后的名字
    showsboxes_face(image, ds, result_path);%把得到的检测框标记在图片上
    disp(['Created ', result_path]);
end
disp('All images processed');
```

通过对 detect_faces.m 的分析，读者对 DPM 人脸检测过程会有一个大概的了解。接下来，我们分析一下 process_face.m 的主要代码。

Code2：process_face.m

```
function [ds, bs] = process(im, model, thresh, nms_threshold)
[ds, bs] = imgdetect(im, model, thresh); %得到初步的人脸检测框 ds
if ~isempty(ds)
   [ds, bs] = clipboxes(im, ds, bs);  %进一步处理人脸检测框
   I = nms(ds, nms_threshold);  %使用 NMS 对 ds 进行处理，得到最终的人脸
检测框
   ds = ds(I,:);
   bs = bs(I,:);
end
```

由 Code1 中的主函数 detect_faces.m 可知，最终得到的人脸检测框通过函数

showsboxes_face 在图片上进行标记，但是这些检测框区域内的图像有的是真实的人脸，有的可能并非人脸。我们称非人脸的检测框为“噪声”，因此，在 showsboxes_face 函数中，我们对每一张图片得到的所有检测框进行判断，如果不是噪声，则在图片上标记出该检测框；否则，忽略该检测框，不标记。

关于如何去噪，这里使用了 GN Stamou 和 I.Pitas 等人提出的方法，步骤如下：

（1）把检测框区域对应的图片切出来，并且把这张图片转换为 HSV 颜色空间。

（2）在 HSV 图像中，如果一个像素点的 0<H<0.15，且 0.2<S<0.6，则称为“像皮肤”（skin-like）。

（3）如果该区域内的 skin-like 个数除以该图的总像素点个数大于 1/4，则认为是真人脸，否则为假人脸。

下面是去噪算法的代码。

Code3：out_false.m

```
function value=out_False(image,x1,y1,x2,y2)
输入:image 为原始图片，x1、y1、x2、y2 为检测框的坐标信息
输出：value=1 代表是人脸；value=0 代表非人脸
    ROI=imcrop(image,[x1,y1,x2-x1,y2-y1]); %把检测框区域切出来保存为 ROI
    hs=rgb2hsv(ROI);  %转换为 HSV 图像
    skin_like= zeros(size(ROI,1),size(ROI,2)) ;
    k=0;
    %对每个像素点遍历，查看是否为 skin-like
    for m=1:size(ROI,1)
        for n=1:size(ROI,2)
if(hs(m,n,1)>0&&hs(m,n,1)<0.15&&hs(m,n,2)>0.2&&hs(m,n,2)<0.6)
                skin_like(m,n)=1;
                k=k+1;
            end
        end
    end
    num=sum(sum(skin_like));  %ROI 中 skin-like 的个数
    if (num/(size(ROI,1)*size(ROI,2)))>1/4
        value=1; %是人脸
    else
        value=0; %非人脸
```

```
    end
end
```

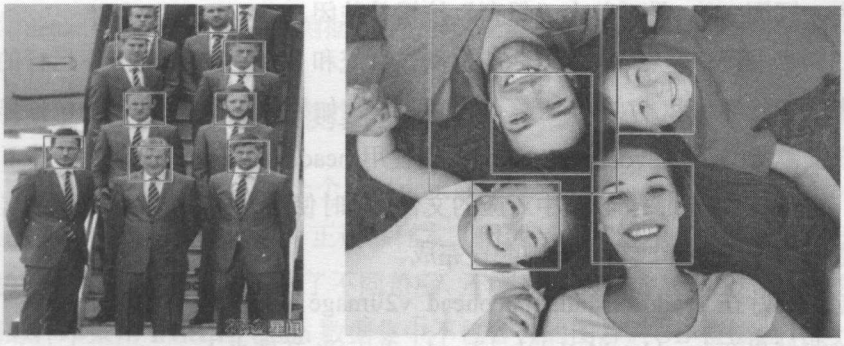
关于该算法的更多细节，可以查看 FaceDetectAlgos\dpmMATLAB 下的所有文件。

### 3.1.3　DPM 人脸检测算法的检测结果

经测试，我们发现 DPM 人脸检测算法的检测质量很好。DPM 人脸检测算法可以检测正脸、侧脸、仰脸、平躺草地等类型的图片，这归功于选择了多角度的训练样本。针对图片尺寸小的情况，检测时间短、效果好；但针对大尺寸的图片，检测时间长，甚至报错。在图 3-4 中，我们呈现了一些有代表性的图片的检测结果。




(a)　(b)　(c)



(d)　(e)

图 3-4　DPM 人脸检测算法的检测结果

## 3.2 LAEO 人脸检测算法

LAEO 人脸检测算法是由 Marin-Jimenez 等人提出的[6]，该算法主要是为了检测视频中的人是否正在看彼此（Looking At Each Other，LAEO）。该算法主要分为三步：①使用 UB（upper-body）检测器，检测出人身体的上半部分，再使用 head detector 在 upper-body 的检测框内检测出人头；②估计人头部的姿势，即偏移角度 yaw 和 pitch；③通过偏移角度判断人们是否正在看彼此（LAEO）。本书介绍的人脸检测程序是这个检测算法的第一部分，Manuel 等人提供了开源的代码和训练模型，我们可以直接使用它进行人脸检测。

### 3.2.1 LAEO 人脸检测算法的使用

LAEO 人脸检测算法，是使用 MATLAB 语言开发的，对应的项目名称为 headdetsMatlab。该项目内包含 leaohead_v2[7]和 voc-release4.01[8]两部分，leaohead_v2 中包含主要函数、测试图片、训练好的模型；voc-release4.01 中包含程序运行过程中需要调用的函数。

在运行该项目之前，首先需要进行配置，具体如下：

（1）首先打开 MATLAB，切换到主函数 test.m 所在的目录。

（2）单击 home→Set Path→ Add to Path with Subfolders，选择 voc-release4.01 文件夹，同图 3-1，最后单击“保存”按钮并关闭。

（3）本书程序是在 Windows 64 位操作系统和 MATLAB 2012b 下运行的。如果读者的运行环境和本实验一致，则配置完成。如果读者使用的是 32 位操作系统或者在文献[8]中下载 voc-release4.01，则使用 headdetsMatlab\change 文件下的.cc 文件，替换掉 voc-release4.01 中对应的文件，同时使用“mex －O xx.cc”对这类.cc 文件进行编译。编译成功后，则配置完成。

检测图片在 headdetsMatlab\laeohead_v2\image 文件夹下。

检测结果在 headdetsMatlab\laeohead_v2\result 文件夹下。

读者可根据实际情况修改检测图片和检测结果的存放路径，具体如下。

修改检测图片的存放路径：在 test.m 中修改 images_folder_path。

```
images_folder_path = 'image\'; %检测图片的存放路径和程序同目录的 image 文件夹
```

修改检测结果的存放路径：在 demoheaddet.m 中修改 out。

```
out=['result/',name(7:length(name)-3),'png']; %检测结果存放路径和检测结果的图片名称
```

选择好路径后，运行 test.m，即可使用 LAEO 人脸检测算法进行测试。

**程序的说明**

本书实现的程序对文献[7]中的 demoheaddet.m 文件进行了修改，因为原程序在检测图片时，如果没有得到检测框，则该程序会报错。此外，为了自动检测一个文件夹下的所有图片，我们创建了 test.m 作为测试的入口，具体如下。

**Code4：test.m**

```
close all;clear all;clc;
images_folder_path = 'image\';  %检测图片存放路径
image_names = dir(fullfile(images_folder_path, '*.jpg'));
for i=1:numel(image_names) %遍历路径下的每一张图片
    image_name = image_names(i).name;
    image_path = fullfile(images_folder_path, image_name);
    demoheaddet(image_path)  %demoheaddet 在图片路径下检测人脸
end
```

在 demoheaddet.m 中，默认的 detthr=−0.82，得分大于 detthr 的框都有可能成为检测框。detthr 值越小，得到的检测框越多，同时噪声也增多。本书中选择 detthr=−0.92。

## 3.2.2　LAEO 人脸检测算法的原理

Manuel 等人首先训练了一个 UB(upper-body)检测器，使用的也是 Felzenswalb 等人的 DPM 目标检测模型[5]。正训练样本采用的是好莱坞电影数据集中标注过的关键帧，这些正训练样本包含了不同角度、不同尺寸的包含人体上半部分的图片；负训练样本采用 INRIA-Person 数据集中不包含人的图片。UB 检测器只包含一个组件。使用这个 UB 检测器单独处理每一帧，随着时间把对应的检测框联合起来，使用 Everingham 等人设计的追踪方法：在不同帧中的同一个检测框被许多 KLT

点连接起来，如果一个追踪中检测框个数小于 20 或者检测得分和小于某个阈值，则认为这个检测框是误报（false-positive）并将之舍弃。

和 UB 检测器类似，LEAO 也使用 Felzenswalb 等人的 DPM 目标检测模型[5]训练一个头部检测器（head detector），使用和 UB 检测器相同的图片进行训练。最后使用 head detector 在 UB 检测框内检测出人的头部对应的坐标位置。

上面分析了 LAEO 人脸检测算法的相关理论，接下来我们结合源代码对该算法进行分析。由 test.m 可知，把存放测试图片路径下的每张图片一一传递给 demoheaddet.m，由 demoheaddet.m 进行人脸检测处理。下面是 demoheaddet.m 的代码，该代码来自文献[7]。

Code5：demoheaddet.m

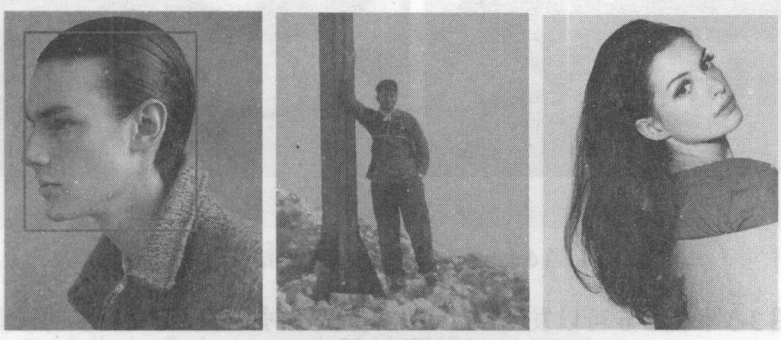
```
function demoheaddet(imagename)
load('head-gen-on-ub-4laeo.mat'); %加载训练模型
%detthr = -0.82;
detthr = -0.92;   %我们修改为-0.92
name=imagename;
img = imread(name);   %读图片
[dets, boxes] = imgdetect(img, model, detthr); %得到检测框矩阵 dets
top = nms(dets, 0.5); %经过 NMS 处理，使每个检测实例只有一个检测框
dets = dets(top,:);
bbox=[];
if  ~isempty(dets)
    bbox = dets(:,[1:4 end-1 end]);
end
%在图片上标记检测框
imshow(img), hold on;
axis on
false_index=[];
if ~isempty(bbox)
    for ii = 1:size(bbox,1)
        value=out_False(img,bbox(ii,1),bbox(ii,2),bbox(ii,3),bbox
        (ii,4)); %对 bbox 进行去噪处理，如果返回 1，则该框是人脸，而 0 代表非人脸
        if value==1
            hr = rectangle('Position', [bbox(ii,1) bbox(ii,2)
bbox(ii,3)-bbox(ii,1) bbox(ii,4)-bbox(ii,2)]);
```

```
            set(hr, 'EdgeColor', [1 0 0]);
            set(hr, 'LineWidth', 3);  %在图片上画检测框
        else
            false_index = [false_index; ii];
        end
    end
end
axis equal;
axis off;
drawnow;
bbox(false_index)=[];
out=['result/',name(7:length(name)-3),'png'];  %检测结果路径和名称
saveas(gcf, out);
close all;clear all;clc;
end
```

通过程序 test.m 和 demoheaddet.m，读者已经对 LAEO 人脸检测算法的步骤有了整体的了解，有关该算法更具体的实现细节，可以查看 FaceDetectAlgos\headdetsMatlab 中的相关文件。

### 3.2.3　LAEO 人脸检测算法的检测结果

和 DPM 相比，LAEO 人脸检测算法的检测效果没有 DPM 好，但是它可以检测尺寸相对大的图片，而 DPM 却不能。下面呈现了一些有代表性的图片的检测结果，如图 3-5 和图 3-6 所示。



（a）　（b）　（c）

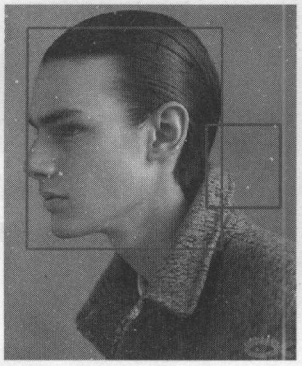
图 3-5　当 detthr=−0.82 时 headdet 算法的检测结果





（d）　　（e）

图 3-5　当 detthr=−0.82 时 headdet 算法的检测结果（续）







（a）　　（b）　　（c）





（d）　　（e）

图 3-6　当 detthr=−0.92 时 headdet 算法的检测结果

由图 3-5 和图 3-6 可以发现，当 detthr 值变小时，可以得到更多的检测框，但检测到更多的噪声（误报）。此外，我们也测试了 detthr=−1 的情况，但结果噪声多、效果不理想。综合上述考虑，我们选择了 detthr=−0.92。

## 3.3　Viola&Jones 人脸检测算法

OpenCV 是一种采用 C/C++编写的开源计算机视觉库，它可以很方便地用于图像处理和分析。OpenCV 视觉库中包含开源的人脸检测程序，本部分主要使用的是 OpenCV 中的 Viola&Jones 人脸检测算法，该算法可以高效地进行人脸检测。

### 3.3.1　Viola&Jones 人脸检测算法的使用

新建 Viola&Jones 人脸检测程序，对应的项目名称为 facedet_opencv。该程序需要在 Visual Studio 平台上安装 OpenCV 才能运行，我们测试使用的是 Visual Studio 2010 和 OpenCV2.4.9。读者首先要确保安装了这两个软件，打开 facedet_opencv.sln 后，还要进行一些配置修改，具体如下。

- 在项目上单击鼠标右键，选择“属性”→“配置属性”→“VC++目录”→“包含目录和库目录”进行修改，修改为自己计算机的 OpenCV 对应的路径。
- 如果你使用的不是 OpenCV2.4.9，则还需要在项目上单击鼠标右键，选择“属性”→“配置属性”→“连接器”→“输入”→“附加依赖项”进行修改。

此时程序就可以正常运行。

测试图片在 facedet_opencv\facedet_opencv\image 文件夹下。

检测结果在 facedet_opencv\facedet_opencv\opencv_result 文件夹下。

读者可根据实际情况，在 facedection.cpp 中修改测试图片和检测结果的存放路径。

```
    string fileFolderPath = "image\\";                    %测试图片的存放路径
    String filename="opencv_result\\"+ (string)fileInfo.name +
"_res.jpg"; %检测结果的存放路径和名称
```

最后运行 facedection.cpp，即可使用该算法进行人脸检测。

### 3.3.2　Viola&Jones 人脸检测算法的原理

Viola&Jones 人脸检测算法主要基于 Viola&Jones 检测器，该检测器是由 Paul

*Viola 和 Michael Jones 在 2001 年发表的[9]，它可以快速、高效地检测人脸。*

Viola&Jones 检测器主要分为三部分：

（1）使用积分通道快速计算图像的特征值。

（2）利用 AdaBoost 分类器筛选特征。

（3）将 AdaBoost 分类器改为级联的分类器，从而快速丢弃非人脸的特征。

OpenCV 视觉库中包含了训练好的脸部级联分类器，我们使用的是：

```
haarcascade_frontalface_alt.xml
```

下面我们结合一些主要代码对该算法进行分析。

Code6 中的主函数 facedection.cpp 的功能就是加载级联分类器、读入图片，然后调用 detectAndDraw 函数进行人脸检测和在图片上标记对应的人脸检测框。

Code7 的 detectAndDraw 函数是实现 Viola&Jones 人脸检测的主要过程，首先对图片进行预处理；然后调用 CascadeClassifier 的 detectMultiScale 方法[10]，即可得到最终的人脸检测框；最后在图片上标记人脸检测框。Code6 和 Code7 均来自 OpenCV 官方网站。

**Code6：facedection.cpp**

```
int main()
{
    Mat image;
    double scale = 1.3;
    CascadeClassifier cascade;//创建级联分类器对象
    string cascadeName = "data/haarcascades/haarcascade_frontalface_
alt.xml";//训练好的人脸分类器
    if( !cascade.load( cascadeName ) ){//加载级联分类器
    cout <<"ERROR: Could not load classifier cascade"<< endl;
    return 0;}
    if( !image.empty() ) {//读取图片数据不能为空
    detectAndDraw( image, cascade, scale);
}
}
```

**Code7：detectAndDraw 函数**

```
void detectAndDraw( Mat& img,CascadeClassifier& cascade,double
scale)
{
```

```
    int i = 0;
    double t = 0;
    vector<Rect> faces;
    const static Scalar colors[] =  { CV_RGB(0,0,255),
    CV_RGB(0,128,255),
    CV_RGB(0,255,255),
    CV_RGB(0,255,0),
    CV_RGB(255,128,0),
    CV_RGB(255,255,0),
    CV_RGB(255,0,0),
    CV_RGB(255,0,255)} ;//用不同的颜色标记不同的人脸
    Mat gray, smallImg( cvRound (img.rows/scale), cvRound(img.cols/
scale), CV_8UC1 );//将图片缩小加快检测速度
    cvtColor( img, gray, CV_BGR2GRAY );//因为用的是类 haar 特征，是基于灰
度图像的，所以这里要转换成灰度图像
    resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
    equalizeHist( smallImg, smallImg );//直方图均衡

    cascade.detectMultiScale( smallImg, faces, 1.1, 2, 0
    //|CV_HAAR_FIND_BIGGEST_OBJECT
    //|CV_HAAR_DO_ROUGH_SEARCH
    |CV_HAAR_SCALE_IMAGE     // CV_HAAR_SCALE_IMAGE 表示使 haar 特征，缩放
的是图片
    ,  Size(30,30) );
    for( vector<Rect>::const_iterator r = faces.begin(); r != faces.
end(); r++, i++ )          //把人脸检测框标记在人脸图片的对应位置上
    {
    Scalar color = colors[i%8];
    rectangle(img, cvPoint(r->x*scale,r->y*scale), cvPoint((r->x+r->
width)*scale,(r->y+r->height)*scale), color,3,8,0);
    }
    }
```

Viola&Jones 图像预处理的步骤如下。

预处理 1：将图片按一定比例缩小。

预处理 2：detectMultiScale 方法需要 CV_8U 矩阵数据作为输入，首先要将图片转换为灰度图像。

预处理 3：使用直方图均衡化进行处理，使级联分类器分析起来更方便。

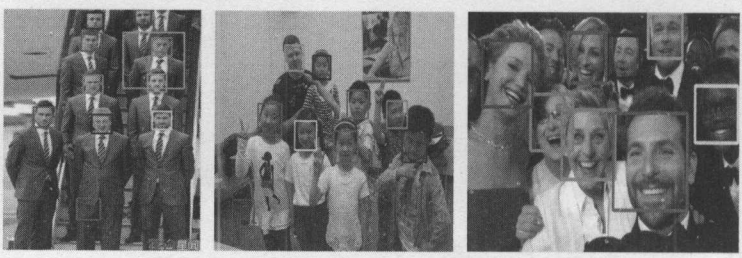
方法 detectMultiScale()的一些主要参数的分析如下。

```
void CascadeClassifier::detectMultiScale(const Mat& image,
vector<Rect>& objects, double scaleFactor=1.1, int minNeighbors=3,
int flags=0, SizeminSize=Size(), Size SizemaxSize =Size())
```

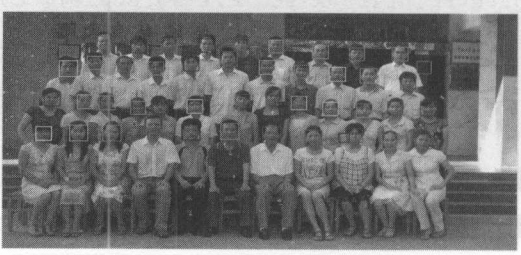
- Mat 矩阵 image——传入的图片，格式是 CV_8U。
- 向量 objects——检测到的所有检测框的位置。
- scaleFactor——每张图片缩放的比例。
- minNeighbors——每个候选检测框应该至少有的邻近元素个数。
- SizeminSize、SizemaxSize 限定目标检测框的范围，即 Size(30,30)。

### 3.3.3 Viola&Jones 人脸检测算法的检测结果

经过大量的测试发现，Viola&Jones 人脸检测算法能够很好地检测正脸，即使检测大尺寸图片，它也可以在短时间内得到很好的检测效果，但是它不能很好地检测非正脸的图片，如图 3-4 的（a）、（b）、（e），这类图片 Viola&Jones 都检测不到。在图 3-7 中，我们呈现了一些有代表性的图片的检测结果。



（a）　　（b）　　（c）



（d）

图 3-7　Viola&Jones 人脸检测算法的检测结果

# 参考文献

[1] The doppia repository. https://bitbucket.org/rodrigob/doppia/src/.

[2] Markus Mathias, Rodrigo Benenson, Marco Pedersoli, Luc Van Gool. Face detection without bells and whistles, 2014.

[3] DPM Software. https://github.com/rbgirshick/voc-dpm/tree/master.

[4] Face Detector. http://markusmathias.bitbucket.org/2014_eccv_face_detection.

[5] Pedro F. Felzenszwalb, Object Detection with Discriminatively Trained Part Based models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2009.

[6] M.J. Marin-Jimenez, A. Zisserman, V. Ferrari. "Here's looking at you, kid." Detecting people looking at each other in videos.British Machine Vision Conference (BMVC), 2011.

[7] LEAO Software. http://www.uco.es/~in1majim/sandbox/headmview/.

[8] VOC Software. http://cs.brown.edu/~pff/latent-release4/.

[9] P Viola. Rapid Object Detection using a Boosted Cascade of Simple Features. CVPR, 2001.

[10] Voila&Jones Face Detector in OpenCV. http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html#cascade- classifier-detectmultiscale.

# 第 4 章

# 基于深度学习的人脸检测算法

本章主要介绍一些较新的、基于深度学习的人脸检测算法，包括 CNN Facial Point Detection 人脸检测算法、DDFD 人脸检测算法，以及不同人脸检测算法的结果融合的技术。

## 4.1 CNN Facial Point Detection 人脸检测算法

CNN Facial Point Detection 人脸检测算法，是由 Yi Sun 等人[1]在 2013 年提出的。该算法主要是实现面部关键点的检测，运行速度快、执行效率高，可以检测侧脸。但如果一张图片被检测到多个人脸，那么在执行面部关键点检测前需要手动处理，比较烦琐。此外，由于 Yi Sun 等人只提供了算法的.exe 可执行文件（只能运行在 Windows 平台上），所以本部分的实验只能在 Windows 操作系统上运行，更多内容可以参考文献[2]。本书提供的程序是调用文献[2]中的 Face detector 程序的可执行文件，并进一步处理。在本部分，我们实验的主要功能是人脸检测，不考虑面部关键点检测。

### 4.1.1　CNN Facial Point Detection 人脸检测算法的使用

该算法需要在 Visual Studio 上配置 OpenCV 视觉库才能运行，本书实现的程序是在 Visual Studio 2010 和 OpenCV2.4.9 上进行测试的，该项目对应的名称为：facedetect_ windows。

打开 facedet_winds.sln 后，还要进行一些配置修改，具体如下。

- 在项目上单击鼠标右键，选择“属性”→“配置属性”→“VC++目录”→“包含目录和库目录”进行修改，修改为自己计算机的 OpenCV 对应的路径。
- 如果你使用的不是 OpenCV2.4.9，则还需要在项目上单击鼠标右键，选择“属性”→“配置属性”→“连接器”→“输入”→“附加依赖项”进行修改。

检测图片在 code_face\image 文件夹下。

检测结果在 result\文件夹下。

读者可根据实际情况，修改存放检测图片和检测结果的路径。

修改存放检测图片的路径：将 code_face/imagelist.txt 的内容修改为检测图片的路径和图片名，第一行指定检测的图片个数，接下来每一行是一个检测图片的路径。

修改存放检测结果的路径：修改 detect_windows.cpp 的 filename。

```
string filename="result\\"+ imagename + "_res.jpg"; %检测结果的路径和名称
```

选择主函数 facedection.cpp，按“F5”键或者单击“运行”按钮，就可以使用 CNN Facial Point Detection 人脸检测算法进行人脸检测。

### 4.1.2　CNN Facial Point Detection 人脸检测算法的原理

Yi Sun 等人提出的算法本质上是检测面部关键点的，检测出左眼中心、右眼中心、鼻子尖、嘴部左角和嘴部右角，使用了三层卷积网络。为了检测效果的准确性和鲁棒性，在第一层使用整个面部区域的所有信息进行特征提取，从而得到较准确的初始估计位置。第二、三层是对初始的估计位置进行调整，达到更高的准确率。它和许多商业软件及最新算法进行对比，获得了很好的检测结果。

CNN Facial Point Detection 人脸检测算法主要包括两部分，第一部分是使用人脸检测器检测人脸，标记出边界框，即文献[2]中的 Face detector；第二部分是

以边界框作为输入检测出面部关键点的位置，即文献[2]中的 Point detector。本书中使用该算法的第一部分进行人脸检测。

我们在主函数 facedection.cpp 中，通过如下语句调用 code_face 的可执行文件，进行人脸检测。

```
    system("code_face\\FacePartDetect.exe code_face\\data code_face\\
imagelist.txt bbox.txt");
```

生成 bbox.txt 文件，该文件中每一行包含：图片名及对应的人脸检测框。每个人脸检测框由四个数据组成，分别对应 left、right、top、bottom。然后读取 bbox.txt 文件，对每张图片进行人脸标记。

### 4.1.3 CNN Facial Point Detection 人脸检测算法的检测结果

经过测试发现，无论检测图片的尺寸大或者小，该算法的运行时间都较短，检测效果好、噪声少，同时可以检测侧脸，但不能很好地检测如图 3-4（e）所示的那类图片。遗憾的是，作者仅公开了可执行程序，而源程序并未开源。在图 4-1 中，我们呈现了一些有代表性的图片的检测结果。

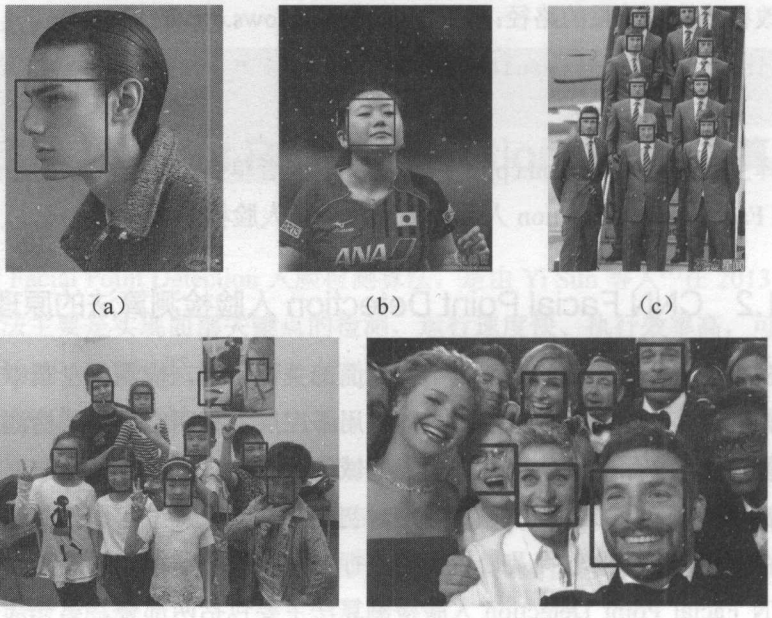



图 4-1　CNN Facial Point Detection 人脸检测算法的检测结果

## 4.2 DDFD 人脸检测算法

DDFD 人脸检测算法是由 Farfade 等人[3]在 2015 年发表的。该算法是一种基于深度学习的人脸检测算法，并可以进行多角度的人脸检测。通常进行多角度人脸检测的算法需要训练大量的模型捕捉各个方向的人脸，还需要对面部关键点或者面部姿势进行注解，这些方法工作量大，比较复杂。DDFD 人脸检测算法只需要一个基于深度卷积神经网络的模型（Deep Dense Face Detector，DDFD），就可以检测各个方向的人脸。

### 4.2.1 DDFD 人脸检测算法的使用

关于 DDFD 人脸检测算法的程序，Farfade 等人没有公开源代码和检测器，本书使用的程序来自文献[4]和文献[5]。

新建一个项目，名称为：Face Detection_CNN-master。该项目包含主要程序[6]和训练好的模型[7]，本书只是利用该算法进行检测。

注意：该算法需要在 Caffe 环境下运行。

读者可根据实际情况，在 test.py 文件中修改检测图片和检测结果的存放路径，具体如下。

检测图片路径：

```
face_detection('face.txt')
```

face.txt 是包含检测图像名称的文档。

检测结果路径：

```
imageFile.save("result/" + str(img_count) + ".jpg")
```

修改路径后，运行 test.py 即可使用 DDFD 人脸检测算法进行人脸检测。

如果读者需要自行训练模型，则可按照如下步骤处理。

(1)运行 image_preprocess.py。该程序是为了对图片进行预处理，使用 AFLW[8]数据库，对于每张图片，如果检测到的人脸与真实的人脸区域重叠面积大于 0.5，则认为是正样本；如果小于 0.3，则认为是负样本。

（2）运行 train.sh。该程序使用 AlexNet[9]在 AFLW 数据库上继续训练，训练后得到的模型为 alexNet__iter_60000.caffemodel[7]。

（3）参考文献[10]将全连接层通过改变参数改成卷积层。

（4）最后运行 test.py。该程序可以得到每张图片的热度图，通过热度图定位人脸区域。

如果读者对检测结果不是很满意，则可以修改 test.py 里面的函数 face_detection 中参数 factor 的值和 delim 中的值，具体如下。

```
def face_detection(imgList):
  img_count = 0
  for imgFile in open(imgList).readlines():
    scales = []
    factor = 0.793700526   %可以修改
    img = Image.open(imgFile.strip())
    min = 0
    max = 0
    if(img.size[0] > img.size[1]):
        min = img.size[1]
    max = img.size[0]
    else:
        min = img.size[0]
    max = img.size[1]
    delim = 2500/max %2500 相当于检测时的最大尺寸，不能太大，否则会内存溢出
    if(delim == 1):
    scales.append(1)
    elif(delim > 1):
        scales.append(delim)
```

## 4.2.2 DDFD 人脸检测算法的原理

使用 AFLW 数据库，对于每张图片来说，如果与检测到的人脸重叠面积大于 0.5，则认为是正样本。所有图片缩放到 227×227 像素。作者通过使用预训练模型 AlexNet Model 和 AFLW 数据库进行调优训练，在调优过程中，设置最大迭代次数为 50K，批处理大小为 128 个，其中 96 个为正样本，32 个为负样本。

作者使用的网络结构与 AlexNet 网络类似，AlexNet 网络包含 8 层，前 5 层

为卷积层，后 3 层为全连接层。作者首先通过重新设置网络参数将最后三层全连接层改成了卷积层，这样就可以通过使用修改后的网络提取任何尺寸的图片的热度图。热度图中的每一个点代表存在面部区域的概率值，对应原来图片中 227×227 像素的区域。然后使用 NMS（非最大值抑制）机制来最终确定人脸区域。最后对图片进行缩放，并分别对缩放得到的图片进行检测，从而得到最终的人脸区域结果。

### 4.2.3　DDFD 人脸检测算法的检测结果

经过大量的测试发现，DDFD 人脸检测算法能够很好地检测多角度人脸，但是检测结果中有很多噪声，根据文献[3]中提到的结论，该算法的检测效果应该是很好的。但是本书测试结果有很多噪声，可能是因为本书程序中训练的模型不好，或者 test.py 参数没有设置好。在图 4-2 中，我们呈现了一些有代表性的图片的检测结果。

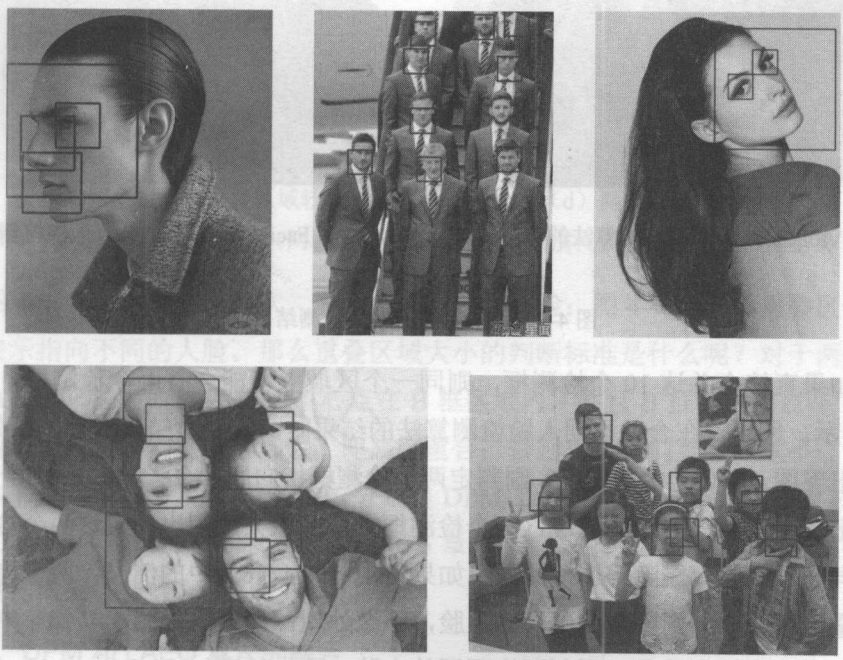
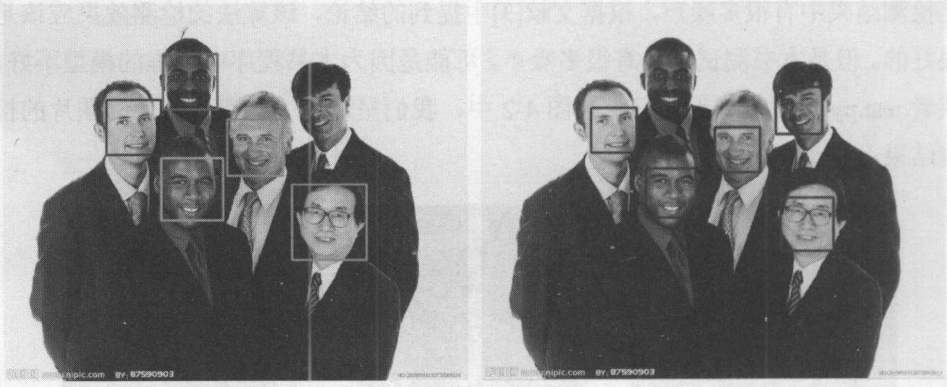


图 4-2　DDFD 人脸检测算法的检测结果

## 4.3 人脸检测算法融合

人脸检测算法融合，这里指将多个算法的检测结果合并，从而尽可能多地检测到所有人脸。多种人脸检测算法的融合，是行之有效的提高人脸检测准确率的方法。如图 4-3 所示，图（a）是 Viola&Jones 人脸检测算法的检测结果，图（b）是 CNN Facial Point Detection 人脸检测算法的检测结果，如果将这两个算法的检测框合并，就可以检测到所有的人脸。



（a）Viola&Jones 人脸检测算法的检测结果　（b）CNN Facial Point Detection 人脸检测算法的检测结果

图 4-3　不同算法的人脸检测结果

如果直接合并这 10 个检测框，则同一个区域上可能会有两个检测框，如图 4-4 所示。所以，在合并不同人脸检测算法的结果时，需要对结果进行判断处理，如果出现图 4-4（a）中的情况，则判定两个检测框指向同一个人脸，只保留一个；如果出现图 4-4（b）中的情况，两个检测框重叠区域大，则也判定这两个检测框指向同一个人脸，删除一个检测框；如果出现图 4-4（c）中的情况，两个检测框的重叠区域较小，则判定为不同的人脸，不做处理；如果出现图 4-4（d）中的情况，两个检测框不重叠，则判断为不同的人脸，不做处理。



（a）两个检测框重合　　（b）两个检测框重叠区域大



（c）两个检测框重叠区域较小　　（d）两个检测框不重叠

图 4-4　合并后检测框的情况（两种不同颜色的检测框分别代表不同算法的检测结果）

图 4-4（b）中重叠区域大，表示指向同一个人脸；图 4-4（c）中重叠区域较小，表示指向不同的人脸。那么重叠区域大小的判断标准是什么呢？对于两个检测框 A、B 来说，如果 A 框的中心点在 B 框区域内，并且 B 框的中心点在 A 框区域内，则表示它们重叠区域大，即高度重合；否则表示它们重叠区域小。

本书介绍了 6 种人脸检测算法，分别是 DPM、LAEO、Viola&Jones、CNN Facial Point Detection、DDFD、Fast R-CNN（第 5 章将介绍）。接下来我们对其中某些算法的检测结果进行融合。

### 1. DPM 和 LAEO 算法的融合

DPM 和 LAEO 算法的融合有两种实现途径：

（1）使用 MATLAB 实现的程序，对应的项目名称为：FaceDetect Algos\dpm_headdet_opencv_Matlab。

在运行之前，需要添加子文件夹 dpm_headdet_opencv_Matlab\voc-release 的一些函数。读者可根据实际情况，在 main.m 文件中修改存放检测图片和检测结果的路径，如下：

```
    out=['result/',[name,'_face.','png']]; %该程序将检测结果存放在当前路径的 result 下
    file_path = 'image'; %该程序将检测图片存放在当前路径的 image 下
```

修改好路径后，单击 main.m 即可测试该合并算法。

（2）使用 C++在 Visual Studio 上实现的程序，对应的项目名称为：FaceDetectAlgos\ dpm_headdet_opencv_C。运行该项目只要在 Visual Studio 上配置好 OpenCV 视觉库即可。

读者可根据实际情况，在 detectFace.cpp 文件中修改存放检测图片和检测结果的路径，如下：

```
    string fileFolderPath ="images\\";  //该程序将检测图片存放在当前路径的 image 下
    string filename="result//"+ (string)fileInfo.name + "_res.jpg"; //该项目将检测结果存放在当前路径的 result 下
```

修改好路径后，单击 detectFace.cpp 即可测试该合并算法。

### 2. CNN Facial Point Detection 和 Viola&Jones 算法的融合

CNN Facial Point Detection 和 Viola&Jones 算法的融合是在 Visual Studio 平台上实现的，对应的项目名称为：FaceDetectAlgos\facedetc_windopencv。

## 参考文献

[1] Y. Sun, X. Wang, X. Tang. Deep Convolutional Network Cascade for Facial Point Detection. In Proceedingsof IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2013.

[2] http://mmlab.ie.cuhk.edu.hk/archive/CNN_FacePoint.htm.

[3] Sachin Sudhakar Farfade, Mohammad Saberian, Li-Jia Li. Multi-view Face Detection Using Deep Convolutional Neural Networks. International Conference on Multimedia Retrieval(ICMR), 2015.

[4] https://github.com/guoyilin/FaceDetection_CNN.

[5] http://blog.csdn.net/guoyilin/article/details/46347271.

[6] Multi-view Face Detection Using Deep Convolutional Neural Networks. https://github.com/guoyilin/FaceDetection_CNN.

[7] 人脸检测之 DDFD. http://blog.csdn.net/qq_14845119/article/details/52564519.

[8] https://lrs.icg.tugraz.at/research/aflw/.

[9] A. Krizhevsky, I. Sutskever, G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Proceedings of NIPS, 2012.

[10] BVLC Repositories, Caffe. http://nbviewer.jupyter.org/github/BVLC/.

# 第 5 章

# 基于 Fast R-CNN 的人脸检测

本章专门介绍一种较新的、快速的人脸检测算法——Fast R-CNN，该算法也基于深度学习技术。

## 5.1 Fast R-CNN 简介

Fast R-CNN（Fast Region-based Convolutional Network Method[1]）是由 Ross Girshick 提出的一种以 R-CNN[2]为基础的目标检测方法。和 R-CNN、SPPnet[3]相比，Fast R-CNN 极大地减少了训练和测试时间，并且提高了检测质量。Fast R-CNN 训练 VGG16 网络，比 R-CNN 快 9 倍，比 SPPnet 快 3 倍；在测试时，比 R-CNN 快 213 倍，比 SPPnet 快 10 倍。Fast R-CNN 在 PASCAL VOC 2012[4]上的 mAP①达到了 66%（R-CNN 的 mAP 为 62%）。

① mAP，即 mean Average Precision（平均准确率），是评价信息检索系统性能的常用指标。

## 5.2 Fast R-CNN 的特点和结构

在 Fast R-CNN 提出以前，基于卷积神经网络的 R-CNN 和 SPPnet 有着很高的目标检测准确率，但是它们仍存在一些缺点和不足。

R-CNN 的缺点：（1）R-CNN 的训练是多阶段的，首先通过 selective search 方法[5]提取 object proposal，然后使用 CNN 网络提出特征，并在 SVM 中分类，最后对边界框回归。(2)R-CNN 训练的时间和空间花费大，使用 VGG16 网络在 GPU 上训练 5000 张图片需要耗时 2.5 天，并且在训练过程中提取的特征需要占用大量的磁盘存储空间。（3）R-CNN 检测时间慢，使用 VGG16 网络在 GPU 上检测一张图片需要 47s。

和 R-CNN 相比，SPPnet 提出了特征共享计算，从而加快了训练和测试的速度。但其仍存在一些缺点：（1）训练是多阶段的。（2）特征需要写入磁盘。（3）不能更新卷积层的权重。

Fast R-CNN 算法弥补了 R-CNN 和 SPPnet 的缺点。它的训练过程是单阶段的。在训练时，特征缓存不需要写进磁盘，并且使用向后传播的方法更新卷积层的权重。图 5-1 所示是 Fast R-CNN 的结构。Fast R-CNN 的网络输入为一张图片和对应的 object proposal 集，网络使用几个卷积网络和最大 pooling 层对整个图像进行处理，从而产生卷积特征 map。对于每个 OP，使用 ROI pooling 层从特征 map 中提取固定长度的特征向量，最后这些特征向量通过一系列全连接层的处理后产生两个输出层：一个是 softmax probability；另一个是边界框回归偏移。Fast R-CNN 不仅提高了训练和测试的速度，而且也增加了检测的准确性。

使用预训练的网络初始化一个 Fast R-CNN 网络需要进行如下操作。

（1）最后一个 max pooling 层被 ROI pooling 层替换。

（2）最后的全连接层和 softmax 被两个输出层替换。

（3）修改网络的输入为：一个图片列表和图片对应的 ROIs 列表。

本书使用 ImageNet 预训练的网络初始化一个侧脸人脸检测模型，下面结合 Fast R-CNN 的源码进行介绍。

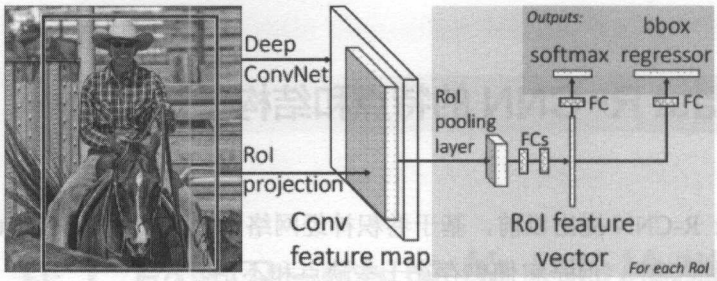



图 5-1 Fast R-CNN 的结构

## 5.3 Fast R-CNN 的使用

Fast R-CNN 需要在基于 Caffe[6]的环境下运行，首先确保已经成功安装和配置 Caffe。本节对 Fast R-CNN 的配置和运行方法进行详细介绍。

### 1. 下载 Fast R-CNN 源码

强烈建议使用终端加命令的方式下载，手动下载会导致内容不完整，并且后期配置时易出错。终端下载的命令为：

```
>>git clone --recursive https://github.com/rbgirshick/fast-rcnn.git
```

本书下载后的项目在 FRCNN 目录下。

### 2. 编译 FRCNN/lib 下的文件

```
>>cd FRCNN/lib
>>make
```

### 3. 编译 FRCNN/caffe-fast-rcnn 下的文件

首先把 caffe/caffe-master 下的 Makefile.config 文件复制到 FRCNN/caffe-fast-rcnn 文件夹内，然后将 Makefile.config 文件中 WITH_PYTHON_LAYER=1 的注释去掉，最后通过如下命令编译。

```
>>cd FRCNN/caffe-fast-rcnn
>>make -j8
>> make pycaffe
>>make matcaffe %可选的，如果执行 MATLAB 的 demo 出错，则需运行该命令
```

4. 下载预训练模型和预选框

```
>>cd FRCNN
>>./data/scripts/fetch_fast_rcnn_models.sh %下载 Fast R-CNN 模型
>>./data/scripts/fetch_selective_search_data.sh %下载 SS 在 VOC 上的预选框
```

5. 运行 demo

可以通过两种方式运行 Fast R-CNN 的 demo。

（1）通过 Python 运行 demo。

```
>>cd FRCNN
>>./tools/demo.py
```

（2）通过 MATLAB 运行 demo。

运行 FRCNN/matlab 下的 fast_rcnn_demo.m。

## 5.4 数据集的预处理

Fast R-CNN 是一个针对多类检测的目标框架，而人脸检测是 2 类检测，即人脸和背景。因此，本实验所用的数据集标注格式和 VOC 不同，并且提取预选框的方法也不同。本节详细介绍一下该实验在训练和测试时所需要提供的数据集格式。

训练时需要提供 3 种文件。

文件 1：待训练的图片名称列表文件，通常是.txt 类型，如下所示。

```
celian/banjiangdianli/001_001.jpg
celian/banjiangdianli/001_002.jpg
celian/banjiangdianli/001_003.jpg
```

文件 2：待训练的图片名称和对应的真实人脸边界框文件，通常是.txt 类型。人脸边界框的坐标格式为：$x_1, y_1, x_2, y_2$，如下所示。

```
celian/banjiangdianli/001_001.jpg 2 146 115 200 171 297 171 339 210
celian/banjiangdianli/001_002.jpg 1 204 50 329 183
celian/banjiangdianli/001_003.jpg 5 254 97 270 116 199 107 241 148
365 85 420 149 315 146 356 183 293 146 312 171
```

文件 3：待训练的图片名称和对应的 object proposal，通常是.mat 类型。目前

有许多算法可以提取图片的 object proposal，如 selective search、EdgeBoxes、GOP 和 LPO、MCG 及 RIGOR 等。Ross 提供的源代码中使用的是 selective search 算法，但本书使用 EdgeBoxes 算法提取 object proposal。

测试时需要提供待测试的图片和对应的 object proposal 矩阵，比如 person1.jpg 和 person1_boxes.mat。下面重点介绍一下如何利用 EdgeBoxes[7]提取训练和测试需要的 object proposal。

## 5.5 EdgeBoxes 的使用

### 1. 下载 EdgeBoxes 源码（https://github.com/pdollar/edges）

下载完成后的项目名称为 edges-master，该项目的运行环境为 MATLAB。

### 2. 下载 Piotr's 工具箱（https://pdollar.github.io/toolbox/）

EdgeBoxes 需要在 Piotr's 工具箱的支持下运行。

### 3. 配置 EdgeBoxes 的运行环境

在 MATLAB 中打开 edges-master，编译一些文件，命令如下。

```
mex private/edgesDetectMex.cpp -outdir private
mex private/edgesNmsMex.cpp    -outdir private
mex private/spDetectMex.cpp    -outdir private
mex private/edgeBoxesMex.cpp   -outdir private
```

编译完成后，可通过如下两种方式将 Piotr's 工具箱添加到该项目中。

方式 1：输入命令 addpath(pwd); savepath。

方式 2：单击 home→setpath→Add with Subfolders，将 Piotr's 的路径添加到项目中。

### 4. 下载 BSDS 500 数据集

BSDS 500 数据集的下载网址为：http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/。

### 5. 运行 demo

至此，EdgeBoxes 配置完成，运行 edgesDemo.m、edgeBoxesDemo.m 和 spDemo.m 等程序，了解如何提取图片的 object proposal。

## 5.6　使用 EdgeBoxes 提取 object proposal

为了对 4034 张图片批量提取 OP（object proposal），本实验编写了一个程序 main.m，该程序可以生成训练和测试网络所要求的 OP 格式，如程序 1 所示。

程序 1：main.m

```
close all;clear all;clc;
fid=fopen('celian/celian_name.txt');
%Demo for Edge Boxes (please see readme.txt first).
%load pre-trained edge detection model and set opts (see edgesDemo.m)
model=load('models/forest/modelBsds'); model=model.model;
model.opts.multiscale=0; model.opts.sharpen=2; model.opts.nThreads=4;

%set up opts for edgeBoxes (see edgeBoxes.m)
opts = edgeBoxes;
opts.alpha = .65;     %step size of sliding window search
opts.beta  = .75;     %nms threshold for object proposal
opts.minScore = .01;  %min score of boxes to detect
opts.maxBoxes = 1e4;  %max number of boxes to detect

images={};
boxes={};

i=0;
while ~feof(fid)
    i=i+1;
    line=fgetl(fid);
    line=regexp(line,' ','split');
    img_name=line(1);
    img_name=img_name{1};
    im = imread(img_name);
    if(ndims(im)==2)
        continue ;
```

```
        end
        box=edgeBoxes(im,model,opts);
        box(:,3)=box(:,1)+box(:,3);
        box(:,4)=box(:,2)+box(:,4);
        box=box(:,1:4);
        %LFW 数据集的保存
        index1=find(img_name=='/');
        index1=index1(length(index1));
        img_name_=img_name(index1+1:end);
        file_name=sprintf('%s_boxes.mat',img_name(1:length(img_name)-4));
        save (file_name,'box')
        images{1,i}=img_name_;
        boxes{1,i}=box;
    end
```

运行 main.m 后，手动选中 images 和 boxes 矩阵，另存为一个矩阵，命名为 edges_celianTrain.mat。该矩阵包含了数据集中所有图片的名称和对应的 OP，供训练使用。

在 main.m 运行的同时，程序会将每张图片提取的 OP 集保存为 mat 矩阵，后缀为_boxes.mat，存放在图片同级目录下，供测试使用。

## 5.7 基于 Fast R-CNN 训练人脸检测网络模型和测试

通过前几节的介绍，读者应该对 Fast R-CNN 有了基本的了解，并且准备好了训练和测试的数据集。本节基于 Fast R-CNN 源码分别介绍训练、测试、评估和优化网络的方法，训练和测试过程主要参考文献[8]。

### 5.7.1 训练阶段

下面以侧脸数据集在 Fast R-CNN 上训练人脸检测网络模型为例，说明 Fast R-CNN 的训练过程。

### 1. 下载预训练网络

首先下载预训练网络，本实验使用的是 ImageNet 模型。

```
>>cd FRCNN
>>./data/scripts/fetch_imagenet_models.sh
```

### 2. 准备数据集

在 FRCNN 项目下，新建一个 DataSet 文件夹存放训练和测试的数据集和对应的文件，目录结构如下：

```
DataSet
   %Celian 训练数据集
   %ImageList_Version_S_AddData.txt 训练图片名称列表
   %ImageList_Version_S_GT_AddData.txt 训练图片名称和对应人脸边界框列表
   %test 测试的数据集和对应的 OP 矩阵
   %test.txt 测试图片名称列表
```

在 5.6 节中使用 EdgeBoxes 方法在训练数据集上提取了对应的 OP 矩阵：edges_celianTrain.mat，将该矩阵放在 FRCNN/data/selective_search_data 路径下。

### 3. 修改程序

FRCNN/lib/datasets 下的文件是与训练和检测相关的程序，因此本实验对 pasca_voc.py、factory.py、imdb.py 和 __init__.py 进行修改。

将 pascal_voc.py 重命名为 wang_detect_face.py，修改的一些主要代码如程序 2 所示。

**程序 2：wang_detect_face.py**

```
class wang_detect_face(datasets.imdb):
    def __init__(self, image_set, devkit_path=None):
        datasets.imdb.__init__(self, image_set)
        self._image_set = image_set
        self._devkit_path = devkit_path
        self._data_path = os.path.join(self._devkit_path)
self._classes = ('__background__','person')
        self._class_to_ind = dict(zip(self.classes,xrange(self.
num_classes)))
 self._image_index = self._load_image_set_index('ImageList_
Version_S_AddData.txt')
```

```
        #Default to roidb handler
        self._roidb_handler = self.selective_search_roidb
        #PASCAL specific config options
        self.config = {'cleanup'  : True,
                       'use_salt' : True,
                       'top_k'    : 2000}
        assert os.path.exists(self._devkit_path), \
               'VOCdevkit path does not exist: {}'.format(self._
devkit_path)
        assert os.path.exists(self._data_path), \
               'Path does not exist: {}'.format(self._data_path)


    def _load_selective_search_roidb(self, gt_roidb):
        filename = os.path.abspath(os.path.join(self.cache_path,
'..','selective_search_data',self.name + '.mat'))
        assert os.path.exists(filename), 'Selective search data not
found at: {}'.format(filename)
        raw_data = sio.loadmat(filename)['boxes'].ravel()
        box_list = []
        for i in xrange(raw_data.shape[0]):
                        box_list.append(raw_data[i][:,:] - 1)
        return self.create_roidb_from_box_list(box_list, gt_roidb)


    def _load_annotation(self):
        gt_roidb = []
        annotationfile = os.path.join(self._data_path, 'ImageList_
Version_S_GT_AddData.txt')
        f = open(annotationfile)
        split_line = f.readline().strip().split()
            num = 1
        while(split_line):
            num_objs = int(split_line[1])
            boxes = np.zeros((num_objs, 4), dtype=np.uint16)
            gt_classes = np.zeros((num_objs), dtype=np.int32)
            overlaps = np.zeros((num_objs, self.num_classes),
dtype=np.float32)
```

```
            for i in range(num_objs):
                x1 = float( split_line[2 + i * 4])
                y1 = float (split_line[3 + i * 4])
                x2 = float (split_line[4 + i * 4])
                y2 = float (split_line[5 + i * 4])
                cls = self._class_to_ind['person']
                boxes[i,:] = [x1, y1, x2, y2]
                gt_classes[i] = cls
                overlaps[i,cls] = 1.0
            overlaps = scipy.sparse.csr_matrix(overlaps)

            gt_roidb.append({'boxes' : boxes, 'gt_classes': gt_
classes, 'gt_overlaps' : overlaps, 'flipped' : False})
            split_line = f.readline().strip().split()
        f.close()
        return gt_roidb
if __name__ == '__main__':
 d = datasets.wang_detect_face('edges_celianTrain', '/home/cz/
Downloads/wwwang/wang/fast-rcnn/DataSet')
    res = d.roidb
    from IPython import embed; embed()
```

程序 3 是 factory.py 修改后的代码，如下所示。

程序 3：factory.py

```
import datasets.wang_detect_face
import numpy as np
__sets = {}
imageset = 'edges_celianTrain'
devkit = '/home/cz/Downloads/wwwang/wang/fast-rcnn/DataSet'
def get_imdb(name):
"""Get an imdb (image database) by name."""
__sets['edges_celianTrain'] = (lambda imageset = imageset, devkit
= devkit: datasets.wang_detect_face(imageset,devkit))
    if not __sets.has_key(name):
            raise KeyError('Unknown dataset: {}'.format(name))
    return __sets[name]()
```

```
def list_imdbs():
"""List all registered imdbs."""
    return __sets.keys()
```

此外，将 wang_detect_face.py、factory.py、imdb.py 和__init__.py 头文件中的 from.pascal_voc import pascal_voc 修改为 from.wang_detect_face import wang_detect_face。

读者可以按照参考文献[8]修改这些文件。

### 4．修改网络

在 FRCNN/data 下的 fast_rcnn_models 文件夹和 imagenet_model 文件夹内，分别有 3 个网络，表 5-1 所示为每个网络模型对应的类型。

表 5-1　网络模型的介绍

| Fast R-CNN | ImageNet | 类型 |
|---|---|---|
| caffenet_fast_rcnn_iter_40000.caffemodel | CaffeNet.v2.caffemodel | 小 |
| vgg_cnn_m_1024_fast_rcnn_iter_40000.caffemodel | VGG_CNN_M_1024.v2.caffemodel | 中 |
| vgg16_fast_rcnn_iter_40000.caffemodel | VGG16.v2.caffemodel | 大 |

本实验使用的网络模型是 imagenet_models 文件夹中的 VGG_CNN_M_1024.v2.caffemodel。在选择好预训练模型后，还需要对网络中的参数进行修改。打开 FRCNN/models/VGG_CNN_M_1024/train.prototxt，进行如下修改。

首先，将 data 层的 num_classes 21 修改为 2（原来是 20 类+背景，本实验为人脸+背景，即 2 类）。

```
layer {
  name: 'data'
  type: 'Python'
  top: 'data'
  top: 'rois'
  top: 'labels'
  top: 'bbox_targets'
  top: 'bbox_loss_weights'
  python_param {
    module: 'roi_data_layer.layer'
```

```
    layer: 'RoIDataLayer'
    param_str: "'num_classes': 2"   //表示人脸和背景
  }
}
```

其次，将 cls_score 层的 num_output 21 修改为 2。

```
layer {
  name: "cls_score"
  type: "InnerProduct"
  bottom: "fc7"
  top: "cls_score"
  inner_product_param {
  num_output: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
  }
}
```

最后，将 bbox_pred 层的 num_output 84 修改为 8（类别数乘以 4）。

```
layer {
  name: "bbox_pred"
  type: "InnerProduct"
  bottom: "fc7"
  top: "bbox_pred"
  inner_product_param {
  num_output: 8
    weight_filler {
      type: "gaussian"
      std: 0.001
    }
  }
}
```

### 5. 运行训练命令

通过前面几个步骤，已经完成了数据的处理、程序的调整和训练网络参数的修改。接下来，使用如下命令训练人脸检测模型。

```
    ./tools/train_net.py --gpu 11 --solver models/VGG_CNN_M_1024_
LOUYIHANG/solver.prototxt --weights data/imagenet_models/VGG_CNN_
M_1024.v2.caffemodel --imdb edges_celianTrain
```

训练结果模型保存在 FRCNN/output/default 下。

### 5.7.2 测试阶段

在进行测试前，首先需要修改测试网络和测试程序。

#### 1. 修改测试网络

将 FRCNN/models/VGG_CNN_M_1024/test.prototxt 文件中 cls_score 层的 num_output 21 修改为 2，bbox_pred 层的 num_output 84 修改为 8。

#### 2. 修改测试程序

本书使用 MATLAB 测试程序，需要修改 FRCNN/matlab/fast_rcnn_demo.m 文件，修改后的文件如程序 4 所示。

程序 4：fast_rcnn_demo.m

```
    function fast_rcnn_demo()
    %Fast R-CNN demo (in matlab).
    [folder, name, ext] = fileparts(mfilename('fullpath'));
    caffe_path = fullfile(folder, '..', 'caffe-fast-rcnn', 'matlab',
'caffe');
    addpath(caffe_path);
    use_gpu = true;
    %You can try other models here:
    def= fullfile(folder,'..','models/VGG_CNN_M_1024/test.prototxt' );
    net = fullfile(folder,'..','output/default/edges_celianTrain/
VGG_CNN_M_1024.v2_iter_40000.caffemodel')
    model = fast_rcnn_load_net(def, net, use_gpu);
    f=fopen('../DataSet/test.txt');  %读入测试的图片集
    fid=fopen('detect_result.txt','a+');   %保存测试结果的文件

    while ~feof(f)
        close all;
        line_str=fgetl(f);
```

```
    str = regexp(line_str, ' ', 'split');
    img_name=str(1);
    img_name=img_name{1};
    fprintf(fid,'%s',img_name);
box=demo(model, img_name(1:end-4), [1], {'person'});
    fprintf(fid,' %d',size(box,1));
    for i=1:size(box,1)
        fprintf(fid,' %d',floor(box(i,1)));
        fprintf(fid,' %d',floor(box(i,2)));
        fprintf(fid,' %d',ceil(box(i,3)));
        fprintf(fid,' %d',ceil(box(i,4)));
    end
    fprintf(fid,'\n');
    fprintf('\n');
end
```

### 3. 运行测试程序

修改完成后，将待测试的图片路径和名称放在 FRCNN/DataSet/test.txt 内，并且每张图片的同级目录下，需要存放对应的 OP 矩阵，以_boxes.mat 为后缀。

最后，执行如下命令：

```
>>cd FRCNN/matlab
>>matlab  %打开 MATLAB
>>运行 fast_rcnn_demo.m
```

运行完成后，检测结果会保存在 detect_result.txt 文件中，如下是本实验的运行结果文件。图 5-2 所示是一些测试图片的检测结果。

运行结果文件：detect_result.txt

```
test/celian/banjiangdianli/001_001.jpg 6 144 114 206 179 303 168
335 215 92 304 115 336 290 238 323 283 289 282 320 329 71 181 104 237
test/celian/banjiangdianli/001_002.jpg 1 200 59 330 187
test/celian/banjiangdianli/001_003.jpg 5 197 109 241 151 369 85 425
141 406 179 470 264 322 143 353 183 297 142 322 173
test/celian/banjiangdianli/001_004.jpg 1 181 23 331 213
```



图 5-2　一些测试图片的检测结果

### 5.7.3　评估阶段

在测试完成后，需要对测试数据集进行评估。本书使用的评估方法和通常的评估方法一致，即若检测框和真实边界框的面积交集/面积并集≥0.5，则认为该检测框检测到的为人脸区域。程序 5 是本书的评估程序。

程序 5：eval_result.m

```
close all;clear all;clc;
f1=fopen('ground_celian.txt'); %真实边界框
f2=fopen('detect_result.txt'); %检测结果文件框
f3=fopen('detect_result_eval_box.txt','a+'); %评估结果文件
GT_num=0; %数据集中真实的人脸数目
detect_num=0; %检测到的人脸数目
detect_face_num=0; %正确检测的数目
```

```
while (~feof(f1)&& ~feof(f2))
    line_str1=fgetl(f1);
    str1 = regexp(line_str1, ' ', 'split');
    img_name1=str1(1);
    img_name1=img_name1{1};

    line_str2=fgetl(f2);
    str2 = regexp(line_str2, ' ', 'split');
    img_name2=str2(1);
    img_name2=img_name2{1};

    if (strcmp(img_name1,img_name2)==0)
        disp('name error');
        break;
    end

    tem_box1=[];
    num1=str2num(char(str1(2)));
    for i=1:num1
        x1  = str2num(char(str1((i-1)*4+3)));
        y1  = str2num(char(str1((i-1)*4+4)));
        x2  =str2num(char(str1((i-1)*4+5)));
        y2  =str2num(char(str1((i-1)*4+6)));
        boxes=[x1 y1 x2 y2];
        tem_box1=[tem_box1;boxes];
    end

    tem_box2=[];
    num2=str2num(char(str2(2)));
    for i=1:num2
        x1  = str2num(char(str2((i-1)*4+3)));
        y1  = str2num(char(str2((i-1)*4+4)));
        x2  = str2num(char(str2((i-1)*4+5)));
        y2  = str2num(char(str2((i-1)*4+6)));
        boxes=[x1 y1 x2 y2];
        tem_box2=[tem_box2;boxes];
```

```
        end

        num3=0;
        for i=1:num2
            score=0;
            for j=1:num1
                detect_box=tem_box2(i,:);
                GT_box=tem_box1(j,:);

                interset_x1=max(detect_box(1,1),GT_box(1,1));
                interset_y1=max(detect_box(1,2),GT_box(1,2));
                interset_x2=min(detect_box(1,3),GT_box(1,3));
                interset_y2=min(detect_box(1,4),GT_box(1,4));
                if(interset_x2<interset_x1 || interset_y2<interset_y1)
                    continue;
                end
                interset_area=(interset_x2-interset_x1)*(interset_y2-
interset_y1);
                GT_area=(GT_box(1,3)-GT_box(1,1))*(GT_box(1,4)-GT_box
(1,2));
                detect_area=(detect_box(1,3)-detect_box(1,1))*(detect_
box(1,4)- detect_box(1,2));
                tem_score=interset_area*1.0/(GT_area+detect_area-
interset_area);
                if(tem_score>=score)
                    score=tem_score;
                end
            end
        if(score>=0.5)
                num3=num3+1;
            end
        end
        GT_num=GT_num+num1;
        detect_num=detect_num+num2;
        detect_face_num=detect_face_num+num3;
        fprintf(f3,'%s',img_name1);
```

```
        fprintf(f3,' %d',num1);
        fprintf(f3,' %d',num2);
        fprintf(f3,' %d',num3);
        fprintf(f3,'\n');
    end
    disp(GT_num);
    disp(detect_num);
    disp(detect_face_num);
```

按照程序 5 评估后，可得到如表 5-2 所示的结果。

表 5-2　检测结果

| 数据集名称 | 数据集中的人脸数目 | 检测到的人脸数目 | 正确检测的数目 | 正确率 | 召回率 |
|---|---|---|---|---|---|
| 侧脸数据集 | 12 674 | 20 495 | 10 944 | 86.4% | 53.4% |

## 5.7.4　优化阶段

由表 5-2 可知，本次训练出的人脸检测模型的正确率只有 86.4%。为了获得更优的人脸检测模型，需要调整 models\VGG_CNN_M_1024\solver.prototxt 文件中的参数，主要调整 base_lr 和 stepsize。

网络参数文件：solver.prototxt

```
train_net: "models/VGG_CNN_M_1024/train.prototxt"
base_lr: 0.001
lr_policy: "step"
gamma: 0.1
stepsize: 30000
display: 20
average_loss: 100
momentum: 0.9
weight_decay: 0.0005
#We disable standard caffe solver snapshotting and implement our
own snapshot
#function
snapshot: 0
#We still use the snapshot prefix, though
snapshot_prefix: "VGG_CNN_M_1024.v2"
#debug_info: true
```

# 参考文献

[1] Girshick R. Fast R-CNN[C]//Proceedings of the IEEE International Conference on Computer Vision, 2015: 1440-1448.

[2] Girshick R, Donahue J, Darrell T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition, 2014: 580-587.

[3] He K, Zhang X, Ren S, et al. Spatial pyramid pooling in deep convolutional networks for visual recognition[C]//European Conference on Computer Vision. Springer International Publishing, 2014: 346-361.

[4] PASCAL VOC. http://host.robots.ox.ac.uk/pascal/VOC/.

[5] Theeuwes J. Stimulus-driven capture and attentional set: selective search for color and visual abrupt onsets[J]. Journal of Experimental Psychology: Human perception and performance, 1994, 20(4): 799.

[6] Caffe. http://caffe.berkeleyvision.org/.

[7] Dollár P, Zitnick C L. Structured forests for fast edge detection[C]//Proceedings of the IEEE International Conference on Computer Vision, 2013: 1841-1848.

[8] http://www.cnblogs.com/louyihang-loves-baiyan/p/4903231.html.

# 第 6 章

# 人脸识别实战

图像和视频处理、深度学习及智能感知设备在当今的智能社会中扮演着核心角色。在智能识别、商品自动分类、无人驾驶汽车、语音识别、刷脸支付、行为识别等应用/研究课题中，图像和视频处理、深度学习技术是需要解决的关键问题。最近几年，很多公司都在预研发以人为中心的视频分析系统。例如，一些公司正在研发针对大型商场的顾客识别与行为分析系统，通过识别、关联某个顾客的购买记录，并通过视频分析识别顾客光临商场的频度、其本次关注商品的种类（如上衣、裤子或皮鞋、运动鞋），可以帮助各专卖店及时识别、抓住重点客户，以增加销售额。人脸识别的另一个重要应用是刷脸支付。上述的两种应用中，都需要高准确度地识别人脸。近年来，人脸识别（主要是正脸识别）技术越来越成熟，已经从学术界应用到工业界，与人们的日常生活也越来越紧密。但由于人群、场景、姿态、光照等各种条件的影响，人脸识别仍然需要更多、更全面、更深入的研究。目前的人脸识别产品，如刷脸考勤机、刷脸验票，都要求采集正脸照片。事实上，基本上所有人脸识别的商业应用，都要求输入正脸照片。而对于侧脸、多角度人脸的识别，学术上也有了一定的研究，但商业化的产品还很少。总之，人脸识别仍然是一个尚未解决的、有挑战的，需要全面、深入研究的课题。本章将详细介绍 5 种人脸识别相关技术。其中 6.1 节和 6.2 节讲述最新的、基于深度学

习的人脸识别技术，6.3 节介绍 OpenCV 中实现的 3 种传统的人脸识别技术，6.4 节对 5 种人脸识别技术进行实验对比分析。

## 6.1 DeepID 算法[1]

DeepID 算法[1]是基于深度卷积神经网络的人脸识别算法，该算法由 Y.Sun 等人在 2014 年提出。在 LFW 数据集上，DeepID 算法具有 99%的识别准确度。在本书中，由于 CelebFaces 和 CelebFaces+数据库并未公开，所以我们在 CASIA-WebFace[2]数据库上进行训练，在 LFW[3]数据集上进行实验验证。LFW 数据集包含 10 575 个人，共有 494 414 张图片，该数据集中每人均有多张图片。本文只基于 Caffe 深度学习平台[4]实现了其中一个 DeepID 深度神经网络，并用于提取人脸的特征。然后，通过计算 LFW 数据集上 6000 对图片特征向量的夹角余弦距离，进行人脸识别的验证，即计算每对人脸（深度特征之间）的夹角余弦距离，并求解最优的夹角余弦距离阈值。

Caffe 平台的安装请参考如下文献：http://caffe.berkeleyvision.org/installation.html 和 http://blog.csdn.net/wangpengfei163/article/details/50488079。

### 6.1.1 DeepID 算法的原理

本书实现的 DeepID 算法流程如下。

（1）首先使用 Y.Sun 等人[5]提出的面部检测软件，检测出所使用的数据集中每张图片中的人脸边框信息，根据人脸边框信息检测面部关键点，包括 2 个眼部中点、鼻尖和 2 个嘴角，共 5 个面部关键点。根据这 5 个关键点首先产生 10 个切片（包括 5 个全局区域和 5 个以关键点为中心的局部区域），如图 6-1（上）所示。每张切片有 3 种尺寸，如图 6-1（下）所示。其又分为灰度图和 RGB 图两种，因此每张检测出来的人脸被处理成了对应的 60 个切片。

1 本节内容来自本书作者所著的《深度学习》一书，由电子工业出版社出版。

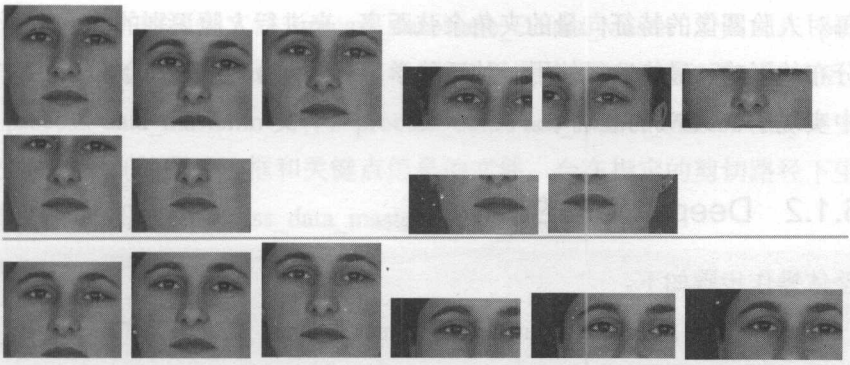


图 6-1　人脸切片（上部：左边五个为全局区域，右边五个为以检测到的五个关键点产生的局部区域；下部：其中两个切片的三个不同的尺寸）

分别对这 60 种切片（共 60 种切片，数据集中每个图像上得到的 60 个切片，分别放到这 60 种切片中）训练深度模型，共训练得到 60 个深度网络结构模型（每种切片对应一种深度模型）。然后，对每个图像的 60 个切片，就可以分别使用这 60 个深度模型对对应切片提取深度特征了。这样，每个切片上可以得到一个 160 维的特征（DeepID 特征向量）。接着，对每个切片进行水平翻转，然后使用相同的深度模型提取特征。最后，每张人脸图片对应的总的 DeepID 特征向量维度为 19 200（160×2×60）。

（2）LFW 数据集包含了 5749 个人，只有 85 个人的图片超过了 15 张，4069 个人只有一张图片。可以看出，LFW 数据集并不适合训练模型（因为在 5749 个人中，只有 1680 个人有两张及以上的图片，故不适合训练机器学习模型）。于是本书采用了 CelebFaces 数据库，随机选取人数的 80%（4349 个人）来训练 DeepID 模型，剩余的 20%用于训练人脸验证模型。在深度特征学习的过程中，使用了切片和该切片对应的水平翻转切片，随机选择每个人 10%的图片来作为训练 DeepID 模型的验证数据。

（3）为了验证论文中的算法和评估结果，选择使用了扩展数据库 CelebFaces+，该数据库包含 10 177 个人，共有 202 599 张图片。随机选取 8700 个人训练 DeepID 模型，剩下的 1477 个人用来训练联合贝叶斯模型做人脸验证。通过使用 5 种不同的尺寸，将每个图像的切片数量提高到 100 个，每个图像的特征向量维度提高到 32 000，然后使用 PCA 将其降维到 150 维。实验结果表明，本书设计的方法在 LFW 数据集中实现了 97.20%的准确率（对 LFW 数据集中的 6000 对图片，分别

计算每对人脸图像的特征向量的夹角余弦距离，来进行人脸识别的验证）。考虑到数据分布的影响，最终选择使用一种迁移学习算法和 10 折交叉验证，在 LFW 数据集中实现了 97.45%的准确率。

### 6.1.2 DeepID 算法的流程

整体操作步骤如下。

（1）运行源码中 process_data/Face_Detect_Master 文件夹下的 process_data_master.m 文件剪切图片。

（2）运行源码中 preprocess_data/Face_Detect_Master 文件夹下的 split_data.m 文件，将数据划分成训练集和验证集。

（3）运行源码中 extract_feature\deepID_webFace\model 文件夹下的 run_deepIDnet.sh 文件，最终会得到后缀名为.caffemodel 的模型文件。

（4）在源码中 extract_feature 文件夹下，getDeepID.m 文件可以实现对指定文件列表中的所有图片进行提取深度特征，classify.m 可以利用提取到的深度特征和夹角余弦距离进行人脸验证。

DeepID 算法的具体实现细节如下。

#### 1．数据预处理

这部分主要是进行数据预处理，代码保存在源码文件夹 code/preprocess_data 下。

（1）我们首先使用 Y.Sun 等人[5]提出的面部检测方法，检测出所使用的数据库中每张图片中包含的人脸边框信息，如图 6-2 所示。

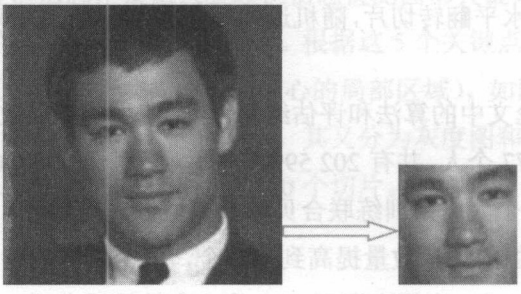


图 6-2　原始图片和裁剪出的面部区域

运行所提供的源码文件 process_data/Face_Detect_Master 工具，在 process_data_master.m 文件中指定原始图片路径、剪切后图片保存路径和图片剪切后尺寸。运行 process_data_master.m 文件，process_data/Face_Detect_Master/Output 文件夹中会生成检测到的图片边框和关键点信息的文件，会在指定的剪切路径下生成对应剪切尺寸的图片。process_data_master.m 文件内容如下：

```
%原始图片路径
root_dir='G:\paper\book\code\face_recognize_deepid\process_data
\CASIA-WebFace-images';
%剪切后图片保存路径
crop_dir='G:\paper\book\code\face_recognize_deepid\process_data
\CASIA-WebFace-cropped-images';
image_list='Output/imageList.list';
all_bbox_list='Output/all_bbox.list';
%是否对剪切后的图片进行缩放
flag_scale=false;
%对剪切后的图片缩放的尺寸
crop_size=[48,48];
process_data(root_dir,crop_dir,image_list,all_bbox_list,flag_scale,
crop_size);

%将数据集划分为训练集和验证集

split_data(crop_dir);

%剪切 LFW 数据集
root_dir='F:\paper\book\code\face_recognize_deepid\extract_feature\
deepID_webFace\data\lfw';
crop_dir='F:\paper\book\code\face_recognize_deepid\extract_feature\
deepID_webFace\data\lfw_cropped_images';
image_list='Output/imageList.list';
all_bbox_list='Output/all_bbox.list';
flag_scale=false;
crop_size=[48,48];
process_data(root_dir,crop_dir,image_list,all_bbox_list,flag_scale,
crop_size);
```

```
    %处理没有检测到人脸的图片，使用原始图片代替
    miss_bbox_fid=fopen('Output/miss_bbox.list','r+');

    while(~feof(miss_bbox_fid))
        line=fgetl(miss_bbox_fid);
        img=imread(fullfile(root_dir,line));
        %得到图片文件名称
        filesep_index=strfind(line,filesep);
        dot_index=strfind(line,'.');
        image_name=line(max(filesep_index)+1:max(dot_index)-1);
        image_extension=line(max(dot_index)+1:end);
        pre_path=line(1:max(filesep_index)-1);

        %创建保存剪切后图片的路径
        crop_image_dir=strcat(crop_dir,filesep,pre_path);
        if ~exist(crop_image_dir,'dir')
            mkdir(crop_image_dir);
        end

        image_full_path=strcat(crop_image_dir,filesep,image_name,'.',
image_extension);
        imwrite(img,image_full_path);
    end

    fclose(miss_bbox_fid);
```

（2）运行源码中 preprocess_data/Face_Detect_Master 文件夹下的 split_data.m 文件，随机选取图片数的 10%作为验证集，剩余图片作为训练集，需要指定图片数据的路径，生成训练集文件 train.txt 和测试集文件 test.txt，文件中每行包含对应的图片路径和标签。split_data.m 文件内容如下：

```
    function split_data(crop_dir)
    crop_dir='G:\paper\book\code\face_recognize_deepid\process_data
\CASIA-WebFace-cropped-images';
    %保存训练集
    train_list_fid=fopen('Output/train.txt','w+');
    %保存验证集
```

```
    val_list_fid=fopen('Output/val.txt','w+');

    label=1;
    sub1_crop_dir=dir(crop_dir);
    for i=1:length(sub1_crop_dir)
        sub1_crop_dir_i=sub1_crop_dir(i);
        if(isequal(sub1_crop_dir_i.name,'.')||isequal(sub1_crop_dir_
i.name,'..'))
            continue;
        end

        %得到图片路径下的所有图片
        images=dir(fullfile(crop_dir,sub1_crop_dir_i.name,'*.jpg'));
        images_num=length(images);
        randNum=randperm(images_num);
        %随机选取图片数的 10%作为验证集
        randIndex=randNum(1:floor(images_num*0.1));

        for m=1:images_num
            image=images(m);
            image_path=fullfile(sub1_crop_dir_i.name,image.name);
            image_path=strrep(image_path,'\','/');

            if find(randIndex==m)
                %将图片路径和标签写入验证集文件
                fprintf(val_list_fid,'%s %d\n',image_path,label);
            else
                %将图片路径和标签写入训练集文件
                fprintf(train_list_fid,'%s %d\n',image_path,label);
            end
        end

        label=label+1;
    end
    fclose(train_list_fid);
    fclose(val_list_fid);

    end
```

## 2. 深度网络结构

我们使用 DeepID 网络结构[1]，主要包括 4 个卷积层、3 个下采样层（Pooling 层）和 2 个全连接层、1 个线性修正单元（ReLU 层）及标记所属类别的输出层。网络最后一个全连接层的输出需要设置为类别数，输入图片尺寸：长方形切片尺寸为 39×31×$k$，正方形切片尺寸为 31×31×$k$，其中 $k$=3 表示为彩色图像，$k$=1 表示为灰度图像。输出为属于每个类别的概率值。我们选择网络结构中的倒数第二个全连接层作为要提取的深度特征层（DeepID 层，维度为 160），由于该层后跟有线性修正单元（ReLU 层），所以提取出的深度特征向量都是非负值。使用的网络结构如图 6-3 所示。

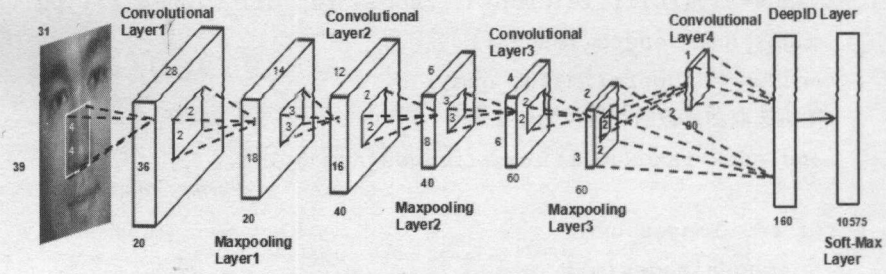



图 6-3　DeepID 网络结构[1]

DeepID 网络结构说明：该网络结构包括 4 个卷积层、3 个下采样层和 2 个全连接层。每一个立方体的维度分别表示输入层、卷积层、下采样层的长度、宽度和高度。内部的小正方形表示卷积核的尺寸。DeepID 层连接了第四个卷积层和第三个下采样层，实现了整体特征和局部特征的结合，称为多尺度网络结构。

我们使用的图片尺寸为 48×48，训练模型所需源码在 extract_feature/deepID_webFace/model 文件夹下，所包含的文件如图 6-4 所示。

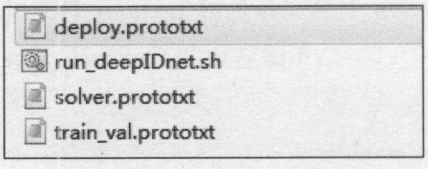



图 6-4　训练深度学习模型需要使用的文件夹下的文件列表

train_val.prototxt 文件定义了 DeepID 网络结构、训练集和验证集，solver.prototxt 文件定义了训练网络所需的网络参数，deploy.prototxt 用于提取深度特征。

在安装有 Caffe 环境的 Ubuntu 系统中，运行 run_deepIDnet.sh 文件，最终会得到后缀名为.caffemodel 的模型文件。run_deepIDnet.sh 文件内容如下：

```
#创建用于训练深度模型的 lmdb 文件

EXAMPLE=/research/res/face_recognize/code/extract_feature/deepID_webFace/model
#数据文件 train.txt 和 val.txt 所在路径
DATA=/research/res/face_recognize/code/extract_feature/deepID_webFace/data
#Caffe 工具所在路径
TOOLS=/stor/caffe/caffe-master/build/tools
#图片所在路径
TRAIN_DATA_ROOT=/research/res/face_recognize/code/extract_feature/deepID_webFace/data/CASIA_WebFace_crop_images/
VAL_DATA_ROOT=/research/res/face_recognize/code/extract_feature/deepID_webFace/data/CASIA_WebFace_crop_images/

#缩放图片尺寸到 48×48
RESIZE=true
if $RESIZE; then
  RESIZE_HEIGHT=48
  RESIZE_WIDTH=48
else
  RESIZE_HEIGHT=0
  RESIZE_WIDTH=0
fi

if [ ! -d "$TRAIN_DATA_ROOT" ]; then
  echo "Error: TRAIN_DATA_ROOT is not a path to a directory: $TRAIN_DATA_ROOT"
  echo "Set the TRAIN_DATA_ROOT variable in create_imagenet.sh to the path" \
       "where the ImageNet training data is stored."
  exit 1
fi

if [ ! -d "$VAL_DATA_ROOT" ]; then
```

```
    echo "Error: VAL_DATA_ROOT is not a path to a directory: $VAL_
DATA_ROOT"
    echo "Set the VAL_DATA_ROOT variable in create_imagenet.sh to the
path" \
         "where the ImageNet validation data is stored."
    exit 1
  fi

  echo "Creating train lmdb..."

  GLOG_logtostderr=1 $TOOLS/convert_imageset \
      --resize_height=$RESIZE_HEIGHT \
      --resize_width=$RESIZE_WIDTH \
      --shuffle \
      $TRAIN_DATA_ROOT \
      $DATA/train.txt \
      $EXAMPLE/deepID_webFace_train_lmdb

  echo "Creating val lmdb..."

  GLOG_logtostderr=1 $TOOLS/convert_imageset \
      --resize_height=$RESIZE_HEIGHT \
      --resize_width=$RESIZE_WIDTH \
      --shuffle \
      $VAL_DATA_ROOT \
      $DATA/val.txt \
      $EXAMPLE/deepID_webFace_val_lmdb

  echo "Done."

  #Compute the mean image from the imagenet training lmdb

  $TOOLS/compute_image_mean $EXAMPLE/deepID_webFace_train_lmdb \
    $EXAMPLE/deepID_webFace_mean.binaryproto

  echo "Done."
```

```
#使用 Caffe 工具开始训练深度模型
/stor/caffe/caffe-master/build/tools/caffe train \
    --solver=/research/res/face_recognize/code/extract_feature/
deepID_webFace/model/solver.prototxt /research/res/face_recognize/
code/extract_feature/deepID_webFace/model/log_solver_deepid_casia.txt
```

说明：源码中涉及的路径问题，请更改为读者自己的对应路径。

### 3. 提取深度特征与人脸验证

上文已经提到，本实验进行人脸验证时，对 LFW 数据集中的 6000 对图片，分别计算每对人脸图像的特征向量的夹角余弦距离，根据该距离的大小判断两个图像是否是同一个人。

本实验选择图 6-3 中的 DeepID 网络结构中的倒数第二个全连接层作为要提取的深度特征层。在本实验中，对 LFW 数据集中的每张图片，我们只使用了一个切片（人脸区域），因此每张图片上最后得到的特征向量维度为 160。

我们穷举不同的阈值，分别获得对应的准确率。最后采用最高准确率对应的阈值，作为本实验的最终结果。

该部分对应的代码包含在源码中 extract_feature 文件夹下，getDeepID.m 文件可以实现对指定文件列表中的所有图片进行提取深度特征，classify.m 可以利用提取到的深度特征和夹角余弦距离进行人脸验证。

getDeepID.m 文件内容如下：

```
function [ deepID] = getDeepID(imagepath,use_gpu,model_def_file,
model_file,featurepath)
%使用训练得到的模型提取图片列表深度特征

disp('extract DeepID feature Begin');
extractFeature=tic;

%重置 Caffe 网络结构
caffe('reset');
%将图片列表输入网络中，提取得到深度特征，输入参数分别为图片列表文件路径、是否
使用 GPU 模式、模型网络文件、模型参数文件
[ori_deepID,list_im,weights]=deepFeature(imagepath,use_gpu,model_
def_file,model_file);
```

```
%所有图片的深度特征，转置为每行对应一张图片的深度特征
deepID=ori_deepID';

%保存特征到 CSV 文件中
for i=1:length(list_im)
    %获得图片文件的名称
    line=list_im{i,1};
    filesep_index=strfind(line,filesep);
    dot_index=strfind(line,'.');
    image_name=line(max(filesep_index)+1:max(dot_index)-1);
    image_extension=line(max(dot_index)+1:end);
    pre_path=line(1:max(filesep_index)-1);

    if ~exist(feature_path,'dir')
        mkdir(feature_path);
    end
    %保存特征到对应的文件中
    saveFeatureToCSV(feature_path,image_name,deepID(i,:));
end

disp('extract DeepID feature Done');

toc(extractFeature);
end
```

classify.m 文件内容如下：

```
function classify(model_def_file,model_file,metric,feature_root,
resultFile)
%输入：网络结构文件（deploy.prototxt）、训练得到的模型参数文件（deepIDnet_
train_iter_450000.caffemodel）、距离方式，保存特征的根目录，保存结果文件
clc;
%将 Caffe 路径添加到 MATLAB 路径中
path('/stor/caffe/caffe-master/matlab/caffe',path);

disp('get the accuracy');
```

```
    datestr(now,31);
    use_gpu=0;
    %待提取特征的图片文件列表
    left_imagepath='/research/res/face_recognize/code/extract_feature/
deepID_webFace/data/left.list';
    right_imagepath='/research/res/face_recognize/code/extract_feature/
deepID_webFace/data/right.list';
    %提取图片列表文件的深度特征
    [left_feature]=getDeepID(left_imagepath,use_gpu,model_def_file,
model_file,strcat(feature_root,'left/'));
    [right_feature]=getDeepID(right_imagepath,use_gpu,model_def_file,
model_file,strcat(feature_root,'right/'));
    %6000 对人脸的标签，同一个人标为 1，否则标为 0
    label_path='/research/res/face_recognize/code/extract_feature/
deepID_webFace/data/label.list';
    label_fid=fopen(label_path,'r+');
    label_cell=textscan(label_fid,'%d');
    label=label_cell{1,1};
    fclose(label_fid);

    image_num=size(left_feature,1);
    %计算 6000 对特征向量之间的距离
    distance = pdist2(left_feature,right_feature,metric);

    %获得每对特征向量之间的距离
    predicts=zeros(image_num,1);
    for i=1:image_num
        predicts(i,1)=distance(i,i);
    end

    %将距离归一化到 0～1，保持跟标签在同一范围
    max_predict=max(predicts);
    min_predict=min(predicts);

    predict_scale=zeros(image_num,1);
    for i=1:image_num
        predict_scale(i,1)=(predicts(i,1)-min_predict)/(max_predict-
```

```
min_predict);
    end
    %设置初始阈值，如果距离大于该阈值，则认为是同一个人，否则不是同一个人
    threshold=0.2;
    [index,acc,thre]=getaccuracy(predict_scale,label,image_num,
threshold,resultFile);

    fprintf('index=%d;threshold=%g;the max acc=%g\n',index,thre,acc);
    end
```

### 6.1.3 DeepID 算法的结果

本部分介绍我们的实验结果。我们在 CASIA-WebFace 数据库中训练 DeepID 模型，进行特征学习；在 LFW 数据集中进行实验验证。由于 CASIA-WebFace 数据库中每人对应的图片并不相等，因此在该数据库中获得的数据是不均衡的，而且存在图片错误归类的情况，会对实验结果造成一定的影响。

**1. 人脸识别数据**

（1）我们在 CASIA-WebFace 数据库上训练 DeepID 模型，进行特征学习。该数据库包含 10 575 个人，共有 494 414 张图片，每人均有多张图片。图 6-5 所示为 CASIA-WebFace 数据库中的一些样例图片。



图 6-5　CASIA-WebFace 数据库中的一些样例图片

（2）我们主要在 LFW 数据集上进行人脸验证，该数据集中一共有 5749 个人，但 4069 个人只有一张对应的图片，而只有 85 个人的图片多于 15 张。图 6-6 所示

为 LFW 数据集中的一些样例图片。



图 6-6　LFW 数据集中的一些样例图片

### 2. 实验细节

在实验中，数据预处理方法使用 6.1.1 节中介绍的方法，生成每个原始图片样本对应的 1 个切片图。使用 6.1.2 节中介绍的方法提取深度特征，获得每个原始图片样本对应的 160 维的特征向量。

通过数据预处理得到数据集，我们随机选取每个人 10%的图片作为训练 DeepID 模型的验证数据，然后使用训练得到的模型提取 LFW 数据集中 6000 对数据的深度特征，计算 6000 对特征向量的夹角余弦距离。通过穷举不同的阈值，得到不同的准确率，我们选择最高准确率作为最终的验证准确率。最终准确率为 0.823 167。得到的准确率并不高，主要原因如下：

（1）CASIA-WebFace 数据库中每人对应的图片并不相等，因此在该数据库中获得的数据是不均衡的，而且存在图片错误归类的情况，会对实验结果造成一定的影响。

（2）该实验中没有使用一些数据增强的方法（增加切片数目的方法）。有兴趣的读者，可以尝试筛选 CASIA-WebFace 数据库，去除错误标记的图片和使用一些数据增强技术（增加切片数目的方法，如每个图像产生 60×2 个切片）来提高最终验证准确率。

我们尝试了不同的参数组合，如逐渐减小学习率、调整批处理大小、测试不同的迭代次数，最终选择较好的参数组合来调优深度学习模型，用于提取深度特征向量。

## 6.2 VGG Face Descriptor 算法

VGG Face Descriptor 算法[6]基于卷积神经网络，其源代码链接为 http://www.robots.ox.ac.uk/~vgg/software/vgg_face/，本书直接使用作者提供的训练模型提取图片特征。特征提取的网络结构保存在 VGG_FACE_16_deploy.prototxt 文件中。该算法使用论文中提供的数据集作为训练特征模型的数据集，在 LFW 和 YFW[7]（Youtube Faces in the Wild）两个数据集上验证最终准确率。

### 6.2.1 VGG Face Descriptor 算法的原理

实现 VGG Face Descriptor 算法的步骤[6]如下。

（1）数据预处理：①首先使用文献[8]中的软件/方法对图像数据集（包括训练图像集、验证图像集、测试图像集三部分）中的每张图像进行人脸检测；②将检测到的每张人脸图像缩放到 227×227 像素；③计算训练图像集的像素均值，将训练图像集和验证图像集的每张图像的所有像素值同时减去该均值；④在使用训练图像集、验证图像集训练深度特征模型之前，以 224×224 像素的尺寸随机剪切（Crop）训练图像集中的每张图片；⑤随机选取训练图像集中 50%的图片进行水平翻转。

（2）设置深度学习的网络结构参数：网络中权重初始化为均值为 0，标准差为 0.01，权重衰减设置为 0.0005，学习率设置为 0.01；然后以 10 的倍数减小学习率以训练模型。最终，选择在验证图像集上预测损失（Loss）最低的一组网络结构参数，作为深度学习模型的最终参数。

（3）特征提取：①对于测试图像集中的每张图像，以 224×224 像素的尺寸，分别从该图像的中心和四个边角进行图像剪切，并对这些切片进行水平翻转，最终每张图像可以得到 10 个 224×224 像素尺寸的切片；②使用（2）中最终训练得到的深度特征模型，对该图像的 10 个切片分别提取特征（每个切片能得到 4096 维的特征向量）；③对这 10 个切片上的 4096 维的特征向量求均值，就得到了该图像对应的 4096 维的特征向量；④为了实现多尺度提取特征，可以首先将图片缩放到 256×256 像素、384×384 像素和 512×512 像素三个尺寸，对这三个尺寸的图像

分别按照①、②、③中的步骤提取到 4096 维的特征向量，最后得到 3 个 4096 维的特征向量，再对这 3 个 4096 维的特征向量求均值，作为该图像最终的特征向量，维度为 4096。

（4）人脸验证：提取 LFW 数据集中每对人脸的深度特征向量，使用欧氏距离进行人脸验证。

### 6.2.2　VGG Face Descriptor 算法的实现

文献[6]的作者公开了 VGG Face Descriptor 的源代码，网址为 http://www.robots.ox.ac.uk/~vgg/software/vgg_face/。源代码中包括特征提取的配置文件 VGG_FACE_16_deploy.prototxt、文献[6]中训练得到的深度学习模型 VGG_FACE.caffemodel、示例图片 ak.jpg 和运行示例文件 matcaffe_demo_vgg_face.m，该文件用于对测试数据集中的图像进行分类。

修改、运行 matcaffe_demo_vgg_face.m 文件，假设要测试 ak.jpg（对应人名为 Aamir_Khan）图像对应的人的标签，运行该文件后即可预测该图像对应的人的标签。该示例 ak.jpg 的预测结果为 3，从源码中提供的 name.txt 文件中找到第 3 行名称是 Aamir_Khan，则可知该预测是正确的。

matcaffe_demo_vgg_face.m 文件由文献[6]提供，其主要内容如下：

```
img = imread('ak.jpg');%读取图片
img = single(img);
averageImage = [129.1863,104.7624,93.5940];%从训练图像集中得到的像素均值
img = cat(3,img(:,:,1)-averageImage(1),...
    img(:,:,2)-averageImage(2),...
    img(:,:,3)-averageImage(3));%图片减去均值
img = img(:, :, [3, 2, 1]); %从 RGB 转换到 BGR
img = permute(img, [2, 1, 3]); %交换宽度和高度

model = 'VGG_FACE_16_deploy.prototxt';%文献[6]中提供的深度学习相关的网络结构
weights = 'VGG_FACE.caffemodel';%文献[6]中提供的深度特征学习模型
caffe('set_mode_cpu');%设置 caffe 使用 CPU 模型运行
caffe('init', model, weights, 'test'); %加载上述网络结构和深度特征学习模型
```

```
    scores = caffe('forward', {img});%将图片 img 输入深度网络模型中，得到属于各类的概率
    scores = scores{1};
    scores = squeeze(scores);
    [~,maxlabel] = max(scores);%用所有预测概率中最高的值对应的标签，预测 img 对应的标签
```

如果仅仅要提取该图像的深度特征（而不需要预测该图像的标签），则只需要去掉 VGG_FACE_16_deploy.prototxt 文件中输出为 2622 的全连接层，即可提取到输入图片 img 的 4096 维的特征向量。

去掉 VGG_FACE_16_deploy.prototxt 文件中输出为 2622 的全连接层之后，再修改 matcaffe_demo_vgg_face.m 文件，去掉它的最后一行，即可得到一个 4096 维的特征向量。

需要强调的是，如果保留文件 VGG_FACE_16_deploy.prototxt 中输出为 2622 的全连接层，如在上面的代码示例中，scores = caffe('forward', {img})得到的是该图像属于各类（标签）的概率；而去掉文件 VGG_FACE_16_deploy.prototxt 中输出为 2622 的全连接层之后，如下面的代码所示，scores = caffe('forward', {img})得到的就是该图像的深度特征向量。

```
    img = imread('ak.png');%读取图片
    img = single(img);
    averageImage = [129.1863,104.7624,93.5940];%训练集得到的均值
    img = cat(3,img(:,:,1)-averageImage(1),...
        img(:,:,2)-averageImage(2),...
        img(:,:,3)-averageImage(3));%图片减去均值
    img = img(:, :, [3, 2, 1]); %从 RGB 转换到 BGR
    img = permute(img, [2, 1, 3]); %交换宽度和高度

    model = 'VGG_FACE_16_deploy.prototxt';%提取特征模型文件
    weights = 'VGG_FACE.caffemodel';%训练得到的深度特征模型
    caffe('set_mode_cpu');%设置 Caffe 使用 CPU 模型运行
    caffe('init', model, weights, 'test'); %加载网络结构和模型
    scores = caffe('forward', {img});%将图片输入到网络中，得到元组形式的深度特征
    scores = scores{1};
    scores = squeeze(scores);%得到图片对应的一列 4096 维的特征向量
```

### 6.2.3　VGG Face Descriptor 算法的结果

不同人脸识别算法在 LFW 数据集上的测试结果对比如表 6-1 所示，对应的 ROC 曲线如图 6-7 所示。

表 6-1　不同人脸识别算法在 LFW 数据集上的测试结果

| 方　法 | 准确率（%） |
| --- | --- |
| DeepID3[9] | 99.47 |
| DeepFace[10] | 97.35 |
| Fisher Vector Faces[11] | 93.10 |
| VGG Face Descriptor | 98.95 |

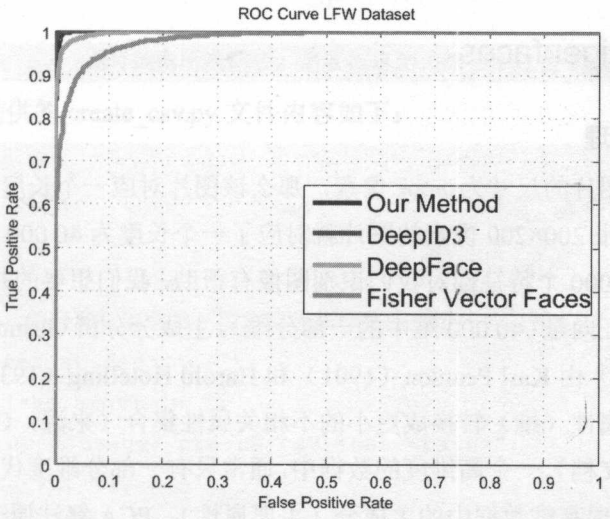



图 6-7　在 LFW 数据集上测试的结果[11]

可以看出，VGG Face Descriptor 算法表现出了较高的性能。其中 DeepID3 算法获得了最高的准确率，但主要是由于该方法对错误标记的测试集进行了纠正，而其他方法并没有做这项工作。

注：（1）6.1 节中使用文献[1]中的网络结构和 CASIA-WebFace 数据库，从初始化参数开始训练，训练得到本书中用来提取深度特征的模型。

（2）6.2 节中作者公开了使用自己的数据集训练得到的模型，本书只讲解了如何使用作者提供的模型提取深度特征，进而进行人脸验证。

## 6.3 OpenCV 中的 3 种人脸识别算法

OpenCV 从 2.4 版本开始支持人脸识别功能，本书所用的版本是 2.6.10，Python 版本为 3.4。OpenCV 主要提供 3 种人脸识别算法，分别是 Eigenfaces[12]、Fisherfaces[13]、Local Binary Patterns Histograms[14]（LBP，局部二值模式直方图）。Eigenfaces 和 Fisherfaces 都是整体特征，而 LBP 则是局部特征，可以降低特征向量维度。OpenCV 人脸识别的官方参考文档链接为：http://docs.opencv.org/2.6.10/modules/contrib/doc/facerec/facerec_tutorial.html。

### 6.3.1 Eigenfaces

#### 1. 算法原理

假设一张图片的尺寸为 $m \times n$ 像素，那么该图片对应一个长度为 $m \times n$ 的特征向量。所以，一张 200×200 像素的图片就对应了一个长度为 40 000 维的特征向量。但并不是这 40 000 个特征都对我们识别图像有帮助，我们想要的是最能代表图片信息的主要特征向量（40 000 维中的一部分维）。主成分分析（Principal Component Analysis，PCA）由 Karl Pearson（1901）和 Harold Hotelling（1933）提出，它将一系列相关的属性（维）转换成更小的不相关属性集合（来源：OpenCV 人脸识别的官方参考文档）。一个高维度的数据中，通常只有一部分维度代表有用的信息，而 PCA 就能发现高维数据中的主成分（主要属性）。PCA 算法原理如下：

（1）计算数据集平均值。

（2）根据数据集平均值计算该数据集的协方差。

（3）求出协方差的特征向量和对应的特征值。

（4）根据特征值降序排列特征向量，前 $k$ 个主成分就是 $k$ 个最大的特征值对应的特征向量。

#### 2. 算法流程

整体操作步骤如下（来源：OpenCV 人脸识别的官方参考文档）。

（1）运行源码 opencv_fr\process_data 文件夹下的 create_csv.py 文件，将图片

路径和标签信息保存到 at.csv 文件中。

（2）编译源码文件夹 opencv_fr 下的 facerec_eigenfaces.cpp 文件，生成可执行文件。

具体步骤如下。

1）数据预处理

（1）使用 AT&T Facedatabase[15]数据库，该数据库包含 40 个人，每个人有 10 张图片，其中某些人的图片在不同时间采集，有微弱的光照变化和面部表情变化。

（2）运行源代码 opencv_fr\process_data 文件夹下的 create_csv.py 文件，将图片路径和标签信息保存到 at.csv 文件中。在使用之前，读者需要修改自己的图片路径。

命令形式：

```
create_csv.py [图片数据所在路径，需要指定到该路径下包含 s*的文件夹]
```

OpenCV 提供的 create_csv.py 文件内容如下：

```
import sys
import os.path

if __name__ == "__main__":
    #保存图片路径和标签信息，图片路径和标签信息用分号隔开
    fid=open('at.csv','w+')
    if len(sys.argv) != 2:
        print ("usage: create_csv <base_path>")
        sys.exit(1)
    #图片数据根目录
    BASE_PATH=sys.argv[1]
    SEPARATOR=";"
    #图片标签从 0 开始
    label = 0
    #遍历该路径下的所有文件夹
    for dirname, dirnames, filenames in os.walk(BASE_PATH):
        for subdirname in dirnames:
            subject_path = os.path.join(dirname, subdirname)
            for filename in os.listdir(subject_path):
                #获得图片文件的绝对路径
```

```
            abs_path = "%s\%s" %(subject_path, filename)
            #将图片路径和标签信息保存到文件 at.csv 中
            fid.write("%s%s%d\n" %(abs_path, SEPARATOR, label))
            print ("%s%s%d" %(abs_path, SEPARATOR, label))
        label = label + 1
fid.close()
```

2）分类识别

（1）编译源码文件夹 opencv_fr 下的 facerec_eigenfaces.cpp 文件，生成的可执行文件命名为 FaceRec_EigenFaces.exe。

（2）使用 FaceRec_EigenFaces.exe 命令：

```
FaceRec_EigenFaces.exe [图片列表文件] [输出图片的保存路径，可选]
```

例：

```
FaceRec_EigenFaces.exe
F:\paper\book\code\opencv_fr\process_data\ at.csv
F:\paper\book\code\opencv_fr\process_data\eigen_result
```

facerec_eigenfaces.cpp 文件是 OpenCV 官方网站提供的源程序，其内容如下：

```
#include "opencv2/core/core.hpp"
#include "opencv2/contrib/contrib.hpp"
#include "opencv2/highgui/highgui.hpp"

#include <iostream>
#include <fstream>
#include <sstream>

using namespace cv;
using namespace std;

static Mat norm_0_255(InputArray _src)
{
    Mat src = _src.getMat();
    //声明归一化图像
    Mat dst;
    switch(src.channels()) {
    case 1:
        cv::normalize(_src, dst, 0, 255, NORM_MINMAX, CV_8UC1);
```

```
            break;
        case 3:
            cv::normalize(_src, dst, 0, 255, NORM_MINMAX, CV_8UC3);
            break;
        default:
            src.copyTo(dst);
            break;
        }
        return dst;
    }
    //读取 CSV 文件
    static void read_csv(const string& filename, vector<Mat>& images,
vector<int>& labels, char separator = ';')
    {
        std::ifstream file(filename.c_str(), ifstream::in);
        if (!file) {
            string error_message = "No valid input file was given, please
check the given filename.";
            CV_Error(CV_StsBadArg, error_message);
        }
        string line, path, classlabel;
        while (getline(file, line)) {
            stringstream liness(line);
            getline(liness, path, separator);
            getline(liness, classlabel);
            if(!path.empty() && !classlabel.empty()) {
                images.push_back(imread(path, 0));
                labels.push_back(atoi(classlabel.c_str()));
            }
        }
    }

    int main(int argc, const char *argv[]) {
        //检查命令行输入参数是否正确
        if (argc < 2) {
            cout << "usage: " << argv[0] << " <csv.ext> <output_folder> "
<< endl;
            exit(1);
```

```
    }
    //图片输出路径
    string output_folder = ".";
    if (argc == 3) {
        output_folder = string(argv[2]);
    }
    //获得 CSV 文件的路径
    string fn_csv = string(argv[1]);
    //保存图片的向量和对应的标签
    vector<Mat> images;
    vector<int> labels;
    //读取图片信息
    try {
        read_csv(fn_csv, images, labels);
    } catch (cv::Exception& e) {
        cerr << "Error opening file \"" << fn_csv << "\". Reason: "
<< e.msg << endl;
        exit(1);
    }
    //如果图片数小于等于 1，则退出
    if(images.size() <= 1) {
        string error_message = "This demo needs at least 2 images to
work. Please add more images to your data set!";
        CV_Error(CV_StsError, error_message);
    }
    //得到第一个图片的高度，需要用来缩放到原来的图片尺寸
    int height = images[0].rows;
    //从数据集中取最后一张图片作为测试，其余的作为训练集
    Mat testSample = images[images.size() - 1];
    int testLabel = labels[labels.size() - 1];
    images.pop_back();
    labels.pop_back();
    //使用图片信息和对应的标签创建 Eigenfaces 模型
    //设置 10 个主成分，可以使用以下语句
    //      cv::createEigenFaceRecognizer(10);
    //
    //设置 10 个主成分和置信阈值，可以使用以下语句
    //
```

```
        //      cv::createEigenFaceRecognizer(10, 123.0);
        //
        //设置所有 EigenFaces 和置信阈值，可以使用以下语句
        //
        //      cv::createEigenFaceRecognizer(0, 123.0);
        //
        Ptr<FaceRecognizer> model = createEigenFaceRecognizer();
        model->train(images, labels);
        //使用训练得到的模型预测测试图片
        int predictedLabel = model->predict(testSample);
        //
        //获得测试图片标签和置信度，可以使用以下语句
        //
        //      int predictedLabel = -1;
        //      double confidence = 0.0;
        //      model->predict(testSample, predictedLabel, confidence);
        //
        string result_message = format("预测类别 = %d / Actual class = %d.",
predictedLabel, testLabel);
        cout << result_message << endl;
        //从 Eigenfaces 模型中获得特征值
        Mat eigenvalues = model->getMat("eigenvalues");
        //从 Eigenfaces 模型中获得特征向量
        Mat W = model->getMat("eigenvectors");
        //获得训练集的平均值
        Mat mean = model->getMat("mean");
        //显示或保存
        if(argc == 2) {
            imshow("mean", norm_0_255(mean.reshape(1,images[0].
rows)));
        } else {
            imwrite(format("%s/mean.png", output_folder.c_str()),
norm_0_255(mean.reshape(1, images[0].rows)));
        }
        //显示或保存 EigenFaces
        for (int i = 0; i < min(10, W.cols); i++) {
            string msg = format("特征值#%d=%.5f",i,eigenvalues.at<double>(i));
            cout << msg << endl;
```

```
            //获得第 i 个特征向量
            Mat ev = W.col(i).clone();
            //调整尺寸到原图尺寸，并归一化到 0～255
            Mat grayscale = norm_0_255(ev.reshape(1, height));
            //使用 JET 模式色图显示图片
            Mat cgrayscale;
            applyColorMap(grayscale, cgrayscale, COLORMAP_JET);
            //显示或保存
            if(argc == 2) {
                imshow(format("eigenface_%d", i), cgrayscale);
            } else {
                imwrite(format("%s/eigenface_%d.png", output_folder.c_
str(), i), norm_0_255(cgrayscale));
            }
        }

        //显示或保存重新构建的图像
        for(int num_components = min(W.cols, 10); num_components <
min(W.cols, 300); num_components+=15) {
            //获得部分特征向量
            Mat evs = Mat(W, Range::all(), Range(0, num_components));
            Mat projection = subspaceProject(evs, mean, images[0].reshape
(1,1));
            Mat reconstruction = subspaceReconstruct(evs, mean, projection);
            //调整尺寸到原图尺寸，并归一化到 0～255
            reconstruction = norm_0_255(reconstruction.reshape(1,
images[0].rows));
            //显示或保存
            if(argc == 2) {
                imshow(format("eigenface_reconstruction_%d", num_
components), reconstruction);
            } else {
                imwrite(format("%s/eigenface_reconstruction_%d.png",
output_folder.c_str(), num_components), reconstruction);
            }
        }
        //如果不保存到文件中，则只显示
        if(argc == 2) {
```

```
        waitKey(0);
    }
    return 0;
}
```

### 3. 算法结果

经测试，该算法能正确预测测试图片的标签。

输出为：测试图片的预测标签，前 10 个特征向量对应的特征值如下所示。

```
预测类别 = 39 / 实际类别 = 39
特征值#0 = 2819989.11572
特征值#1 = 2062664.43241
特征值#2 = 1096663.64953
特征值#3 = 894183.32702
特征值#4 = 818967.15747
特征值#5 = 537972.44192
特征值#6 = 392438.01898
特征值#7 = 372326.54572
特征值#8 = 313616.90351
特征值#9 = 288570.19349
```

使用了 JET 模式色图的 10 个 Eigenfaces 如图 6-8 所示，从这些图中我们可以看出，Eigenfaces 不仅对面部特征进行了编码，而且对图片中的光照也进行了编码（第四张图中的右侧光，第五张图中的左侧光）。

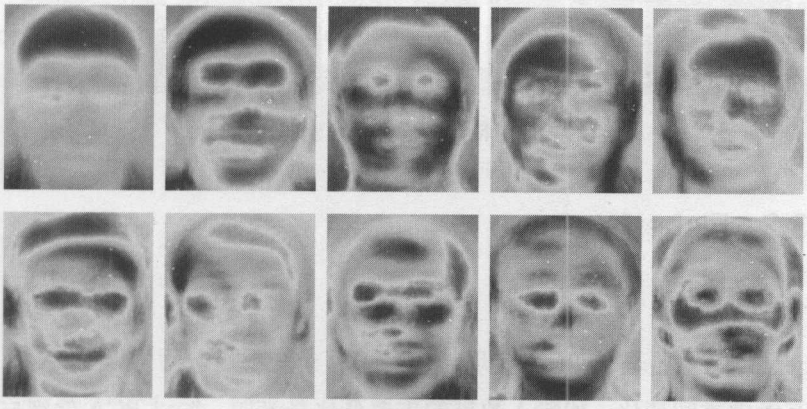


图 6-8　使用了 JET 模式色图的 10 个 Eigenfaces

使用特征向量重新构建的人脸图像如图 6-9 所示。重新构建一个较好的人脸图像需要多个特征向量，从图 6-9 中可以看出，10 个特征向量不足以重新构建人脸图像。50 个特征向量可能足够重新构建人脸图像，对于 AT&T Facedatabase 来说，300 个特征向量可以重新构建一个较好的人脸图像（来源：OpenCV 人脸识别的官方参考文档）。



图 6-9　使用特征向量重新构建的人脸图像

## 6.3.2　Fisherfaces

### 1. 算法原理

Eigenfaces 主要基于 PCA 实现，但是仍有一些不足，因为该 PCA 是无监督的，没有考虑类间变化，如输入数据的主要变化是光照，经过 PCA 处理后，该数据不再具有可区分信息（来源：OpenCV 人脸识别的官方参考文档）。

线性鉴别分析[16]（Linear Discriminant Analysis，LDA）主要是处理特定类的降维，最大化类间的离散度、最小化类内的离散度。Fisherfaces 主要基于 LDA 实现。

**2．算法流程**

整体操作步骤如下。

（1）运行源码 opencv_fr\process_data 文件夹下的 create_csv.py 文件，将图片路径和标签信息保存到 at.csv 文件中。

（2）编译源码文件夹 opencv_fr 下的 facerec_fisherfaces.cpp 文件，生成可执行文件。

具体步骤如下。

1）数据预处理

同 6.3.1 节。

2）分类识别

编译源码文件夹 opencv_fr 下的 facerec_fisherfaces.cpp 文件，生成 FaceRec_FisherFaces.exe 可执行文件。

FaceRec_FisherFaces.exe 命令：

```
FaceRec_FisherFaces.exe [图片列表文件] [输出图片的保存路径，可选]
```

例：

```
FaceRec_FisherFaces.exe
F:\paper\book\code\opencv_fr\process_ data\at.csv
F:\ paper\book\code\opencv_fr\process_data\fisher_result
```

facerec_fisherfaces.cpp 文件是 OpenCV 官方网站提供的源程序，其内容如下：

```
#include "opencv2/core/core.hpp"
#include "opencv2/contrib/contrib.hpp"
#include "opencv2/highgui/highgui.hpp"

#include <iostream>
#include <fstream>
#include <sstream>

using namespace cv;
using namespace std;
```

```
static Mat norm_0_255(InputArray _src)
{
    Mat src = _src.getMat();
    //声明归一化图像
    Mat dst;
    switch(src.channels()) {
    case 1:
        cv::normalize(_src, dst, 0, 255, NORM_MINMAX, CV_8UC1);
        break;
    case 3:
        cv::normalize(_src, dst, 0, 255, NORM_MINMAX, CV_8UC3);
        break;
    default:
        src.copyTo(dst);
        break;
    }
    return dst;
}
//读取CSV文件
static void read_csv(const string& filename, vector<Mat>& images,
vector<int>& labels, char separator = ';')
{
    std::ifstream file(filename.c_str(), ifstream::in);
    if (!file) {
        string error_message = "No valid input file was given, please
check the given filename.";
        CV_Error(CV_StsBadArg, error_message);
    }
    string line, path, classlabel;
    while (getline(file, line)) {
        stringstream liness(line);
        getline(liness, path, separator);
        getline(liness, classlabel);
        if(!path.empty() && !classlabel.empty()) {
            images.push_back(imread(path, 0));
            labels.push_back(atoi(classlabel.c_str()));
        }
```

```
    }
}

int main(int argc, const char *argv[])
{
    //检查输入参数是否正确
    if (argc < 2) {
        cout << "usage: " << argv[0] << " <csv.ext> <output_folder>
" << endl;
        exit(1);
    }
    string output_folder = ".";
    if (argc == 3) {
        output_folder = string(argv[2]);
    }
    //获得输入参数中的 CSV 文件
    string fn_csv = string(argv[1]);
    // These vectors hold the images and corresponding labels.
    vector<Mat> images;
    vector<int> labels;
    //读取 CSV 文件
    try {
        read_csv(fn_csv, images, labels);
    } catch (cv::Exception& e) {
        cerr << "Error opening file \"" << fn_csv << "\". Reason: "
<< e.msg << endl;
        exit(1);
    }
    //如果图片数小于等于 1，则退出
    if(images.size() <= 1) {
        string error_message = "This demo needs at least 2 images to
work. Please add more images to your data set!";
        CV_Error(CV_StsError, error_message);
    }
    //得到第一个图片的高度，我们需要用来缩放到原来的图片尺寸
    int height = images[0].rows;
    //从数据集中取最后一张图片作为测试，其余的作为训练集
```

```
        Mat testSample = images[images.size() - 1];
        int testLabel = labels[labels.size() - 1];
        images.pop_back();
        labels.pop_back();
        //使用图片信息和对应的标签创建 Fisherfaces 模型
        //设置 10 个 Fisherfaces，可以使用以下语句
        //
        //      cv::createFisherFaceRecognizer(10);
        //
        //设置所有 Fisherfaces 和置信度阈值，可以使用以下语句
        //
        //      cv::createFisherFaceRecognizer(0, 123.0);
        //
        Ptr<FaceRecognizer> model = createFisherFaceRecognizer();
        model->train(images, labels);
        //使用训练得到的模型预测测试图片
        int predictedLabel = model->predict(testSample);
        //
        //获得测试图片标签和置信度，可以使用以下语句
        //
        //      int predictedLabel = -1;
        //      double confidence = 0.0;
        //      model->predict(testSample, predictedLabel, confidence);
        //
        string result_message = format("预测类别 = %d / Actual class = %d.",
predictedLabel, testLabel);
        cout << result_message << endl;
        //从 Fisherfaces 模型中获得特征值
        Mat eigenvalues = model->getMat("eigenvalues");
        //从 Fisherfaces 模型中获得特征向量
        Mat W = model->getMat("eigenvectors");
        //获得训练集的平均值
        Mat mean = model->getMat("mean");
        //显示或保存
        if(argc == 2) {
            imshow("mean", norm_0_255(mean.reshape(1, images[0].rows)));
        } else {
```

```
            imwrite(format("%s/mean.png", output_folder.c_str()), norm_
0_255(mean.reshape(1, images[0].rows)));
        }
        //显示或保存最多 16 个 Fisherfaces:
        for (int i = 0; i < min(16, W.cols); i++) {
            string msg = format("特征值#%d = %.5f", i, eigenvalues.at
<double>(i));
            cout << msg << endl;
            //获得第 i 个特征向量
            Mat ev = W.col(i).clone();
            //调整尺寸到原图尺寸，并归一化到 0～255
            Mat grayscale = norm_0_255(ev.reshape(1, height));
            //使用 BONE 模式色图显示图片
            Mat cgrayscale;
            applyColorMap(grayscale, cgrayscale, COLORMAP_BONE);
            //显示或保存
            if(argc == 2) {
                imshow(format("fisherface_%d", i), cgrayscale);
            } else {
                imwrite(format("%s/fisherface_%d.png", output_folder.
c_str(), i), norm_0_255(cgrayscale));
            }
        }
        //显示或保存重新构建的图像
        for(int num_component = 0; num_component < min(16, W.cols);
num_component++) {
            //获得指定特征向量
            Mat ev = W.col(num_component);
            Mat projection = subspaceProject(ev, mean, images[0].reshape
(1,1));
            Mat reconstruction = subspaceReconstruct(ev, mean,
projection);
            //调整尺寸到原图尺寸，并归一化到 0～255
            reconstruction = norm_0_255(reconstruction.reshape(1,
images[0].rows));
            //显示或保存
            if(argc == 2) {
```

```
            imshow(format("fisherface_reconstruction_%d", num_
component), reconstruction);
        } else {
            imwrite(format("%s/fisherface_reconstruction_%d.png",
output_folder.c_str(), num_component), reconstruction);
        }
    }
    //如果不保存到文件中，则只显示
    if(argc == 2) {
        waitKey(0);
    }
    return 0;
}
```

### 3. 算法结果

经测试，本算法能正确预测测试图片的标签。

输出为：测试图片的预测标签，此处只输出了前 16 个类别的特征值，如下所示。

```
预测类别 = 39 / 实际类别 = 39
特征值#0 = 46343770.45151
特征值#1 = 12526.75725
特征值#2 = 2156.94929
特征值#3 = 1202.72304
特征值#4 = 643.75567
特征值#5 = 415.09794
特征值#6 = 366.10602
特征值#7 = 216.88946
特征值#8 = 160.48730
特征值#9 = 134.74498
特征值#10 = 87.77821
特征值#11 = 76.46510
特征值#12 = 49.96221
特征值#13 = 47.58930
特征值#14 = 42.38283
特征值#15 = 34.96763
```

输出 16 个类的 Fisherface 如图 6-10 所示，输出 16 个类的 Fisherface 重新构建后的图像如图 6-11 所示。

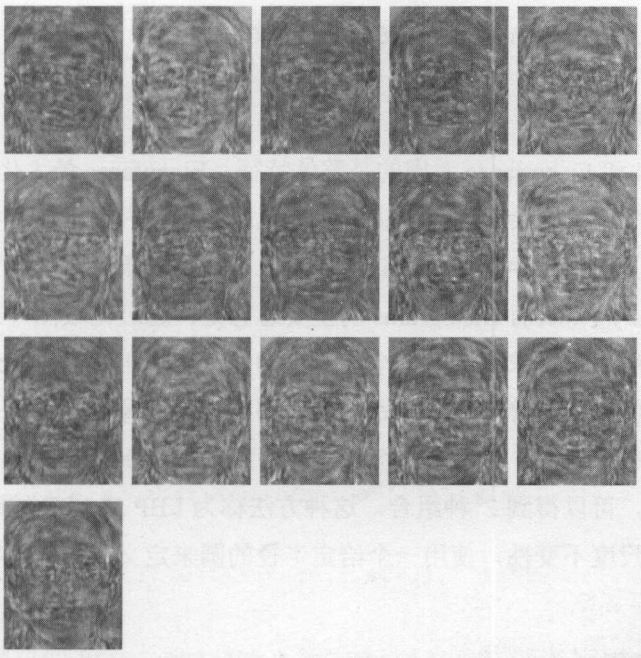


图 6-10　16 个类的 Fisherface



图 6-11　16 个类的 Fisherface 重新构建后的图像

### 6.3.3 Local Binary Patterns Histograms

#### 1. 算法原理

Eigenfaces 和 Fisherfaces 考虑的是整体特征，Eigenfaces 最大化整体离散度，由于数据内部变化可能会导致分类识别，如光照变化等；为了保留类间可区分信息，于是就有了基于 LDA 实现的 Fisherfaces，在特定场景下，它优于 Eigenfaces。

Local Binary Patterns Histograms 的主要思想是：通过比较图片中像素和与它相邻的像素对局部进行求和。取一个像素作为中心，对相邻的像素进行比较，如果中心像素的值大于或等于相邻像素的值，则把该相邻像素标为 1，否则标为 0。对于图片中的每个像素，会得到一个二进制序列，如 00001111。一个中心像素周围的 8 个像素，可以得到 $2^8$ 种组合，这种方法称为 LBP 或 LBP 编码。

为了具有尺度不变性，使用一个给定半径的圆来定义近邻，这种方法称为扩展 LBP。

#### 2. 算法流程

整体操作步骤如下。

（1）运行源码 opencv_fr\process_data 文件夹下的 create_csv.py 文件，将图片路径和标签信息保存到 at.csv 文件中。

（2）编译源码文件夹 opencv_fr 下的 facerec_lbph.cpp 文件，生成可执行文件。

具体步骤如下。

1）数据预处理

同 6.3.1 节。

2）分类识别

编译源码文件夹 opencv_fr 下的 facerec_lbph.cpp 文件，假设生成 FaceRec_LBPH.exe 可执行文件。

FaceRec_LBPH.exe 命令：

```
FaceRec_LBPH.exe [图片列表文件] [输出图片的保存路径，可选]
```

例：

```
FaceRec_LBPH.exe  F:\paper\book\code\opencv_fr\process_data\at. csv
```

facerec_lbph.cpp 文件是 OpenCV 官方网站提供的源程序，其内容如下：

```
#include "opencv2/core/core.hpp"
#include "opencv2/contrib/contrib.hpp"
#include "opencv2/highgui/highgui.hpp"

#include <iostream>
#include <fstream>
#include <sstream>

using namespace cv;
using namespace std;
//读取 CSV 文件
static void read_csv(const string& filename, vector<Mat>& images,
vector<int>& labels, char separator = ';')
{
    std::ifstream file(filename.c_str(), ifstream::in);
    if (!file) {
        string error_message = "No valid input file was given, please
check the given filename.";
        CV_Error(CV_StsBadArg, error_message);
    }
    string line, path, classlabel;
    while (getline(file, line)) {
        stringstream liness(line);
        getline(liness, path, separator);
        getline(liness, classlabel);
        if(!path.empty() && !classlabel.empty()) {
            images.push_back(imread(path, 0));
            labels.push_back(atoi(classlabel.c_str()));
        }
    }
}

int main(int argc, const char *argv[])
{
    //检查命令行输入参数是否正确
    if (argc != 2) {
```

```
        cout << "usage: " << argv[0] << " <csv.ext>" << endl;
        exit(1);
    }
    //获得 CSV 文件的路径
    string fn_csv = string(argv[1]);
    //保存图片对应的向量和标签
    vector<Mat> images;
    vector<int> labels;
    //读取图片信息
    try {
        read_csv(fn_csv, images, labels);
    } catch (cv::Exception& e) {
        cerr << "Error opening file \"" << fn_csv << "\". Reason: "
<< e.msg << endl;
        exit(1);
    }
    //如果图片数小于等于 1，则退出
    if(images.size() <= 1) {
        string error_message = "This demo needs at least 2 images to
work. Please add more images to your data set!";
        CV_Error(CV_StsError, error_message);
    }
    //得到第一个图片的高度，我们需要用来缩放到原来的图片尺寸
    int height = images[0].rows;
    //从数据集中取最后一个图片作为测试，其余的作为训练集
    Mat testSample = images[images.size() - 1];
    int testLabel = labels[labels.size() - 1];
    images.pop_back();
    labels.pop_back();
    //使用图片信息和对应的标签创建 LBPH 模型
    //LBPHFaceRecognizer 使用了扩展 LBP
    //默认值
    //
    //      radius = 1
    //      neighbors = 8
    //      grid_x = 8
```

```
        //      grid_y = 8
        //
        //设置半径为 2，邻域为 16，可以使用以下语句
        //
        //      cv::createLBPHFaceRecognizer(2, 16);
        //
        //设置置信度阈值和其他参数，可以使用以下语句
        //
        //      cv::createLBPHFaceRecognizer(1,8,8,8,123.0)
        //
        Ptr<FaceRecognizer> model = createLBPHFaceRecognizer();
        model->train(images, labels);
        //使用训练得到的模型预测测试图片
        int predictedLabel = model->predict(testSample);
        //
        //获得测试图片标签和置信度，可以使用以下语句
        //
        //      int predictedLabel = -1;
        //      double confidence = 0.0;
        //      model->predict(testSample, predictedLabel, confidence);
        //
        string result_message = format("预测类别 = %d / Actual class = %d.",
predictedLabel, testLabel);
        cout << result_message << endl;
        //在没有重新训练模型的前提下，设置阈值为 0
        model->set("threshold", 0.0);
        //设置阈值为 0 后，预测标签为-1，因为没有小于 0 的距离值
        predictedLabel = model->predict(testSample);
        cout << "预测类别 = " << predictedLabel << endl;
        //为了保证效率，LBP 图片没有保存在模型中
        cout << "模型信息:" << endl;
        string model_info = format("\tLBPH(radius=%i, neighbors=%i,
grid_x=%i, grid_y=%i, threshold=%.2f)",
                model->getInt("radius"),
                model->getInt("neighbors"),
                model->getInt("grid_x"),
```

```
            model->getInt("grid_y"),
            model->getDouble("threshold"));
    cout << model_info << endl;
    //获得模型直方图
    vector<Mat> histograms = model->getMatVector("histograms");
    //输出直方图的个数
    cout << "直方图个数: " << histograms[0].total() << endl;
    return 0;
}
```

### 3. 算法结果

经测试，该算法能正确预测测试图片中的标签。

输出为：测试图片的预测标签，模型信息如下所示。

```
预测类别 = 39 / 实际类别 = 39
模型信息:
        LBPH(radius=1, neighbors=8, grid_x=8, grid_y=8, threshold=0.00)
直方图个数: 16384
```

## 6.4 人脸识别算法对比分析

本节主要对 6.1、6.2、6.3 节中涉及的 5 种人脸识别算法在公开数据集上进行对比分析。

表 6-2 展示了 DeepID 算法和 VGG Face Descriptor 算法在 LFW 数据集上，针对 6000 对人脸图像的测试结果。可以看到，DeepID 算法优于 VGG Face Descriptor 算法。而在 LFW 数据集上，Eigenfaces 算法只能达到 60%的识别准确度。

表 6-2 DeepID 算法和 VGG Face Descriptor 算法在 LFW 数据集上的测试结果

| 方　法 | 验证准确率（%） |
| --- | --- |
| DeepID 算法 | 82.3167 |
| VGG Face Descriptor 算法 | 49.9833 |

在表 6-3 中，我们使用 6.3 节中 OpenCV 提供的 3 种人脸识别算法在 AT&T

Facedatabase 数据库上进行测试，随机选取每个人图片数的 20%作为测试集，剩余的作为训练集。

表 6-3　OpenCV 提供的 3 种人脸识别算法在 AT&T Facedatabase 数据库上的测试结果

| 方　法 | 验证准确率（%） |
| --- | --- |
| Eigenfaces | 98.68 |
| Fisherfaces | 93.42 |
| Local Binary Patterns Histograms | 96.05 |

从表 6-3 中可以看出，Eigenfaces、Fisherfaces 和 Local Binary Patterns Histograms 在 AT&T Facedatabase 数据库上都表现出了较高的准确率，其中 Eigenfaces 获得了最佳的准确度。

## 6.5　小结

人脸识别包括三个基本步骤，具体如下。

第一个步骤是从图像中检测人脸边框，所用的是人脸检测技术。而人脸检测技术有多种，可以是基于肤色的，也可以是基于人脸特征点（如眼睛）的方法，等等。这一部分内容在本书第 3 章和第 4 章已经给出。

人脸识别的第二个步骤是人脸区域的特征提取。根据算法原理和关注点的不同，人脸的特征可以有很多种。深度学习特征是一种特征；还有 Gabor、SIFT、HOG 等多种特征提取方法（将在后续章节中讲述），用于提取人脸的特征；通过 PCA、LDA 等特征降维的方法得到的特征，也可以作为人脸的特征；人脸面部的特征点信息，可以选取 5 个面部特征点（双眼中心、鼻子、两个嘴角），或 20 个特征点（包括眉梢、鼻端、下巴、双眼中心、鼻子、两个嘴角等信息），或 30 个乃至更多的特征点，每个特征点的位置和像素信息都可以用于表征人脸的特征。学术界已有人脸特征点自动定位算法。另外，人脸的轮廓也是一种重要特征。人脸特征提取的方法，以及最后所使用的特征，对人脸识别的准确度具有关键作用。

人脸识别的第三个步骤是训练识别人脸特征与人的标签对应关系的分类模型。经过第二步，训练图像集中的每个图像都可以用一个特征向量来表示（如一

个 4096 维的特征向量），最后再加上该图像的标签（表示是哪个人的人脸）。一个图像对应一行类似于上面所属的记录，如果有 $m$ 个训练图像，就得到 $m$ 行这样的记录矩阵。然后，在该矩阵上，使用 SVM、KNN、Softmax Regression、Logistic Regression 等方法训练分类模型。利用最后得到的模型，预测测试图像集中每个图像对应的人的标签。

在上述三个步骤中，第二个步骤是最为重要的，也是决定人脸识别准确度的关键。近年来，随着大数据、深度学习技术的发展，人们越来越多地使用深度学习来提取人脸特征。例如，本章 6.1 节和 6.2 节中的 DeepID 和 VGG Face Descriptor 人脸识别算法，都代表当今世界上最新的人脸识别技术。但这并不表明深度学习就可以完全替代其他特征提取方法。事实上，最新的研究表明，将深度特征与“传统”特征（如 Gabor、LBP、HOG 等）恰当地结合起来，可以取得更佳的识别准确度；而在一些应用中，不使用深度特征也许能取得更高的识别准确度。深度学习技术通常是以大数据为前提的，即需要海量的、有标注（标签）的数据，才能取得优于“传统”特征提取方法的识别准确度；对于小规模的数据集，使用深度学习技术得到的特征，未必能取得最高的识别准确度。作者认为，在实际应用中，人们迫切需要一个自动的“特征提取方法工厂”。该“工厂”中集成了所有重要、常见的特征提取方法，并结合具体的图像数据，自动验证、测试不同的特征提取方法，自动选择一种或多种特征提取方法的结合，以达到最佳的分类准确度。目前，尚没有类似自动的“特征提取方法工厂”。目前常见的做法是，人们基于手动测试，手动地尝试、验证若干种不同的特征提取方法或一些方法的结合。很明显，这一过程有相当的主观性和偶然因素，即并不能确保人们最后使用的一个或多个方法能取得最优的识别准确度。而自动的“特征提取方法工厂”面临的主要挑战之一是计算能力，例如，对于一个普通的图像数据集，如果使用深度学习方法，测试一组参数可能需要运行 24 小时甚至更长的时间。因此，进行大规模的测试以求解最优参数将耗费大量的计算和时间。在很多情况下，这是计算条件或时间不允许的。譬如，据本书作者了解，为了进行深度学习方面的研究和开发，通常需要不少于 40 块 Nvidia Tesla K40 的 GPU 集群，以减少深度学习调参所需的时间。而穷举地将深度学习方法与其他“传统”特征提取方法结合起来以寻找最适合给定图像集的、能取得最佳识别准确度的一个或多个特征提取方法，也将耗费更多的时间，可能是数周，甚至是数月的时间。尽管如此，本书所述

的自动的“特征提取方法工厂”一定是未来人脸识别、图像处理研究和开发的重点发展方向。

# 参考文献


[1] Sun Y, Wang X, Tang X. Deep Learning Face Representation from Predicting 10,000 Classes[C]// 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE Computer Society, 2014:1891-1898.

[2] Dong Yi, Zhen Lei, Shengcai Liao, Stan Z. Li. Learning Face Representation from Scratch. arXiv preprint arXiv:1411.7923. 2014.

[3] Huang G B, Mattar M, Berg T, et al. Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments[J]. Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition, 2007.

[4] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T.Darrell. Caffe: Convolutional architecture for fast feature.

[5] Sun Y, Wang X, Tang X. Deep Convolutional Network Cascade for Facial Point Detection[C]// Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on. IEEE, 2013:3476-3483.

[6] O. M. Parkhi, A. Vedaldi, A. Zisserman. Deep Face Recognition British Machine Vision Conference, 2015.

[7] L. Wolf, Tal. Hassner, I. Maoz. Face recognition in unconstrained videos with matched background similarity. In Proc. CVPR, 2011.

[8] M. Mathias, R. Benenson, M. Pedersoli, L. Van Gool. Face detection without bells and whistles. In Proc. ECCV, 2014.

[9] Y. Sun, L. Ding, X. Wang, X. Tang. DeepID3: Face recognition with very deep neural networks. CoRR, abs/1502.00873, 2015.

[10] Y. Taigman, M. Yang, M. Ranzato, L. Wolf. DeepFace: Closing the gap to human-level performance in face verification. In Proc. CVPR, 2014.

[11] K. Simonyan, A. Vedaldi, A. Zisserman. Learning local feature descriptors using

convex optimisation. IEEE PAMI, 2014.

[12] Matthew Turk, Alex Pentland. Eigenfaces for recognition. Journal of cognitive neuroscience, 3(1):71-86, 1991.

[13] Peter N. Belhumeur, João P Hespanha, David Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 19(7):711-720, 1997.

[14] Timo Ahonen, Abdenour Hadid, Matti Pietikäinen. Face recognition with local binary patterns. In Computer vision-eccv 2004, pages 469 - 481. Springer, 2004.

[15] Samaria F S, Harter A C. Parameterisation of a Stochastic Model for Human Face Identification[C]// Applications of Computer Vision, Proceedings of the Second IEEE Workshop on. IEEE, 1994:138-142.

[16] Fisher, R. A. The use of multiple measurements in taxonomic problems. Annals Eugen. 7 (1936), 179-188.

# 第 7 章

# 人脸检索实践

随着科技的飞速发展，数码相机、手机、数字摄像机等信息设备已经非常普及，这些设备产生的图像规模越来越大。而对这些图像的有效管理和高效检索成为一个亟须解决的问题。例如，在公安应用中，工作人员调出监控视频并检索犯罪嫌疑人相关的视频和活动轨迹。面对多个摄像头录制的监控视频，办案人员用眼睛逐个查看这些视频是目前的通用做法。然而，这将耗费大量的时间，因此迫切需要高效、自动检索犯罪嫌疑人相关视频片段的技术。因此，图像检索，尤其是人脸检索，是一个非常重要的研究领域。

## 7.1 人脸检索简介

图像检索，亦称以图搜图。输入一张图像，算法或程序高效、快速地返回图像库中与输入图像相似的图像。如果以一个向量来表示一张图像，在检索的过程中，需要依据这些向量做图像间（输入图像与图像库中的每张图像）的相似性检索/匹配。

对于人脸图像，其处理过程如下：第一步，检测每张图像中的人脸，即人脸

检测；第二步，对每张图像中的人脸区域进行特征提取并用这些特征组成的向量表示该图像；第三步，对输入的查询图像（Query Image）做与第二步相同的处理；第四步，用第三步中得到的输入图像的人脸特征向量，与图像库中每一个人脸的特征向量进行相似性计算。最终，返回图像库中与输入图像相似的所有图像。

在第一步中，我们可以使用多种人脸检测算法，如 Viola&Jones 人脸检测等算法，在本书第 3 章中已经讲述过这类算法。

在第二步和第三步中，可以使用 SIFT、HOG（Histogram of Oriented Gradient）、Gabor、Deep Learning 等多种不同的算法，也可以使用 PCA 降维、感知哈希（PHash）算法、DHash 等算法。

在第四步中，主要使用两种算法，一种是向量之间直接比对的 Naïve 查询处理算法，另一种是基于 KD-Tree 结构的高效查询处理算法，尤其是 FLANN（Fast Approximate Nearest Neighbor Search Library）中实现的 KD-Tree 算法/数据结构。

对于图像库中的每张图像，都需要经过第一步和第二步的处理。而对于用户查询的每张图像，都需要经过第三步和第四步的处理。也就是说，不管是图像库中的每张图像，还是输入的查询图像，都需要经过图像特征提取的步骤，这里需要使用上面提到的 SIFT、HOG、Gabor、Deep Learning 等算法。

特征提取和输入图像的查询处理算法，是相对独立的两个步骤。在本章中，PHash、DHash、PCA、SIFT、Gabor、HOG、Deep Learning 都属于特征提取/降维的算法；而 Naïve 查询处理算法和 KD-Tree 查询处理算法是特征提取之后，进行查询处理的算法。

下面我们将分别介绍各种特征提取的算法及查询处理算法，每个算法都将分别介绍其使用、原理、算法实现、实验数据、实验结果及分析。

## 7.2 计算人脸相似度的方法

本章中，我们主要使用欧氏距离和余弦相似度的度量方法作为计算图像间相似性的依据。欧氏距离和余弦相似度的最大区别是：前者强调点的思想，后者注重线的思想。下面我们来详细了解一下它们的定义及公式。

### 7.2.1　欧氏距离

欧氏距离（Euclidean Distance）是一种常用的距离定义，指 $n$ 维空间中两个点之间的实际距离，也就是说，我们将高维空间向量都视为一个高维空间上的点，用欧氏距离即可求得两点间的距离。文献[1]给出了 $n$ 维空间中的欧氏距离公式。

$n$ 维欧氏空间是一个点集，它的每个点 $X$ 或向量 $\boldsymbol{x}$ 可以表示为$(x[1], x[2],\cdots,x[n])$，其中 $x[i](i=1,2,\cdots,n)$是实数，称为 $X$ 的第 $i$ 个坐标。

两个点 $A=(a[1],a[2],\cdots,a[n])$ 和 $B=(b[1],b[2],\cdots,b[n])$之间的距离 $\rho(A,B)$定义为下面的公式：

$$\rho(A,B)=\sqrt{[\Sigma(a[i]-b[i])^2]}\quad (i=1,2,\cdots,n)$$

向量 $\boldsymbol{x}=(x[1],x[2],\cdots,x[n])$ 的自然长度$|x|$定义为下面的公式：

$$|x|=\sqrt{(x[1]^2+x[2]^2+\cdots+x[n]^2)}$$

计算欧氏距离的代码如下：

```
double calculateEuDis(Mat targetDescriptor,Mat baseDescriptor)
{
    double sumDescriptor = 0;
    for(int c=0; c<baseDescriptor.cols; c++)
    {
        double numB = abs(baseDescriptor.at<float>(0,c));
        double numT = abs(targetDescriptor.at<float>(0,c));

        sumDescriptor += pow(numB-numT,2);
    }
    double simility = sqrt(sumDescriptor);

    return simility; //simility的值越小，越相似
}
```

### 7.2.2　余弦相似度

余弦相似度是利用求两个向量之间的余弦相似度来衡量两个向量之间的夹

角。余弦相似值越接近 1，夹角越小，它们的方向就越吻合，则认为它们越相似；反之，则越不相似。文献[2]给出了 $n$ 维空间上的余弦相似度公式。

对于 $n$ 维空间，设向量 $\boldsymbol{A}=(A_1,A_2,\cdots,A_n)$，$\boldsymbol{B}=(B_1,B_2,\cdots,B_n)$。其中分子是两个向量的内积或称点积，分母是两个向量的欧几里得长度的乘积。

$$\cos\theta\frac{\sum_1^n(A_i\times B_i)}{\sqrt{\sum_1^n A_i^2}\times\sqrt{\sum_1^n B_i^2}}$$

本章算法中我们计算余弦相似度的代码如下：

```
double calculateCosEuDis(Mat targetDescriptor, Mat baseDescriptor)
{
    double sumDescriptor = 0;
    double squSumB=0;
    double squSumT=0;
    double innerPro=0;
    for(int c=0; c<baseDescriptor.cols; c++)
    {
        Double numB = abs(baseDescriptor.at<float>(0,c));
        double numT = abs(targetDescriptor.at<float>(0,c));
        squSumB=squSumB+numB*numB;
        squSumT=squSumT+numT*numT;
        innerPro=innerPro+numB*numT;
    }
    double modB=sqrt(squSumB);
    double modT=sqrt(squSumT);
    double simility = innerPro/(modB*modT);
    return simility;   //simility 的值越大，越相似
}
```

除欧氏距离和余弦相似度外，汉明距离（Hamming Distance）也是一种常用的计算两个向量之间相似度的方法。汉明距离的思想是，一一比较每个位（属性）上，两个向量的值是否相同，如果不相同，则汉明距离加 1。最后得到的汉明距离越小，表明两个向量之间的相似度越高。

## 7.3　查询处理算法

本章中我们主要使用两种查询处理算法，分别是 Naïve 和 KD-Tree。

在 Naïve 查询处理算法中，分别计算查询图像的特征向量与图像库中的每张图像的特征向量之间的（欧氏距离或余弦）相似度。最后，返回图像库中与输入图像的特征最相似的若干张图像。

KD-Tree 算法是对图像库中的每张图像的特征，建立一个 KD-Tree 结构，以实现对输入的查询点的快速查询处理。KD-Tree 的相关原理将在本章 7.9 节中给出。

Naïve 查询处理算法和 KD-Tree 查询处理算法的主要区别在于检索相似特征向量的运行时间。这两个查询处理算法与特征提取的步骤是独立的。给定用特征向量表征的图像（从图像中提取特征之后），运用 Naïve 查询处理算法和 KD-Tree 查询处理算法，都可以检索图像库中与查询图像最相似的那些图像。

## 7.4　评价人脸检索结果的标准

查准率和查全率主要用于信息检索领域，它们一般作为反映检索效果的重要指标。

查准率，亦称精度，衡量检索结果中噪声所占的比重。公式为：查准率=(检索出的相关信息量/检索出的信息总量)×100%。

查全率，亦称召回率，衡量检索结果中成功召回的比例。公式为：查全率=(检索出的相关信息量/系统中的相关信息总量)×100%。

误报（False Positive）个数，即返回的结果中，与查询图像无关的图像的个数。

## 7.5　PHash 算法

PHash 算法（Perceptual Hash algorithm），即感知哈希算法，计算基于低频的

均值哈希。它对每一张图像生成一个指纹字符串，进而通过该字符串比较不同图片对应的指纹。若结果越接近，就说明图片相似度越高。可应用于以图搜图的应用中。下面是我们基于 OpenCV 实现的算法及相关分析。本算法思想见参考文献[3]。

### 7.5.1 PHash 算法的使用

在 Windows7 环境下，使用 Microsoft Visual Studio 2010 开发环境，配置 OpenCV2.4.9 及以上版本。

该算法的项目文件名称是“phashProject.sln”，位于“..\phash”路径下。具体操作步骤如下。

（1）打开 PHash→phashProject.sln，然后打开源文件 phashproject.cpp。

（2）输入训练数据集路径和格式替换时可在主函数下面两行代码中修改：

```
string fileFolderPath = "image150x200";
string fileExtension = "jpg";
```

（3）修改输入测试数据的路径时，应在以下代码中修改：

```
sprintf(model_fileName, "image150x200\\S001-002.jpg");
```

（4）输出数据在与项目 phashproject.cpp 文件同一路径下的 hanming-distance.txt 文件中。

按照以上步骤提示执行 phashproject.cpp 文件即可。

### 7.5.2 PHash 算法原理

Phash 算法的基本原理是：首先将图片转换为灰度图，再将图片调整到 32×32 像素的尺寸并通过 DCT 变换，取左上角 8×8 像素的区域。然后，计算这 64 个像素的灰度平均值，将每个像素的灰度值与平均值进行比较，大于平均值的记为 1，否则记为 0，从而得到 64 位哈希值。

### 7.5.3 PHash 算法实现

PHash 算法的具体计算步骤如下。

（1）将输入图片转换成灰度图：

```
if(image1.channels()==3)
{
   cvtColor(image1,image1,CV_BGR2GRAY);
   image = Mat_<double>(image1);
}
else
   image=Mat_<double>(image1);
```

若是 RGB 彩色图像，则将其转换成灰度图像。

（2）缩小图片的尺寸为 32×32 像素：

```
resize(image, image, Size(32, 32));
//缩小图片的尺寸为 32×32 像素
```

这样做的目的是便于 DCT 变换，节省运行时间。

（3）DCT（Discrete Cosine Transform）变换：

```
        dct(image, imageDct);//DCT 变换
```

对于一般的图像而言，大多数 DCT 系数值都接近于 0，因此去掉这些系数不会对重建图像的质量产生较大的影响。这样，利用 DCT 进行图像压缩可以节省大量的存储空间和运行时间。

（4）缩小 DCT，保留左上角 8×8 像素的区域：

```
double Sum=0;
for(int i=0;i<8;i++)
    for(int j=0;j<8;j++)
        Sum += imageDct.at<double>(i, j);
```

一般来说，变换后的 DCT 系数值较大的会集中在区域的左上部，即低频分量都集中在左上部。所以，我们只取左上角 8×8 像素的区域。

（5）计算 8×8 像素区域的平均值，进行像素值比较并求 hash 值：

```
double Average=Sum/64;
for(int i = 0 ; i < 8 ; i++)
{
    uchar* inputImageData = image.ptr<uchar>(i);
    for(int j = 0 ; j < 8 ; j++)
```

```
        {
            if(imageDct.at<double>(i, j)>Average)
                phashcode.at<char>(i,j)=1;
            else
                phashcode.at<char>(i,j)=0;
        }
    }
```

计算最终 DCT 变换后的所有像素点（8×8）的平均值，并对每一个像素点遍历，若像素点大于平均值，则记为 1；反之，则记为 0。组合成 64 个信息位作为该图像的信息指纹，顺序保持一致性即可。

（6）指纹（hash 值）匹配：

```
    int d=0;
    for(int c=0;c<64;c++)
    {
     if( baseDescriptor.at<float>(0,c)!=targetDescriptor.at<float>
(0,c))
        d++;
    }
```

计算两幅图像哈希值之间的汉明距离（两个相同长度的哈希值之间的汉明距离）。显然，汉明距离越大，表示图片越不相似。

### 7.5.4 PHash 算法的实验数据、实验结果及分析

#### 1. 实验数据

实验的输入图像集在项目路径下，若需要替换图像集，则可在主函数的前两行进行修改。其中，第一行表示图像的路径名，第二行表示图像的类型。

输入查询图像的路径：

```
    sprintf(model_fileName, "image150x200\\S001-002.jpg");
    //查询图像
```

#### 2. 实验结果

查询图像如图 7-1 所示。



图 7-1　查询图像（1）

检索出的部分相似图像如图 7-2 所示。



图 7-2　检索出的部分相似图像（1）

实验结果保存在与 phashproject.cpp 同一路径下的 hanming-distance.txt 和 fileName.txt 文件内。其中，fileName.txt 表示图像集的图像名，hanming-distance.txt 中每一行表示对应图像与查询图像的汉明距离。部分结果如表 7-1 所示。

表 7-1　部分实验结果（1）

| | | |
|---|---|---|
| image150x200\S001-001(1).jpg | 1 | yes |
| image150x200\S001-001.jpg | 1 | yes |
| image150x200\S001-002.jpg | 0 | yes |
| image150x200\S001-003.jpg | 2 | yes |
| image150x200\S001-004.jpg | 2 | yes |
| image150x200\S001-005.jpg | 3 | yes |
| image150x200\S001-006.jpg | 4 | yes |
| image150x200\S001-007.jpg | 3 | yes |
| image150x200\S001-008.jpg | 1 | yes |
| image150x200\S001-009.jpg | 4 | yes |

续表

| | | |
|---|---|---|
| image150x200\S001-010.jpg | 4 | yes |
| image150x200\S001-011.jpg | 4 | yes |
| image150x200\S001-012.jpg | 7 | |
| image150x200\S001-013.jpg | 5 | yes |
| image150x200\S001-014.jpg | 5 | yes |
| image150x200\S001-015.jpg | 5 | yes |
| image150x200\S001-016.jpg | 5 | yes |
| image150x200\S001-017.jpg | 4 | yes |
| image150x200\S001-018.jpg | 6 | |
| image150x200\S001-019.jpg | 3 | yes |
| image150x200\S001-020.jpg | 3 | yes |
| image150x200\S001-021.jpg | 4 | yes |
| image150x200\S001-022.jpg | 3 | yes |
| image150x200\S001-023.jpg | 3 | yes |
| image150x200\S001-024.jpg | 3 | yes |
| image150x200\S001-025.jpg | 4 | yes |
| image150x200\S001-026.jpg | 5 | yes |
| image150x200\S001-027.jpg | 5 | yes |
| image150x200\S001-028.jpg | 4 | yes |
| image150x200\S001-029.jpg | 4 | yes |
| image150x200\S001-030.jpg | 3 | yes |
| image150x200\S001-031.jpg | 3 | yes |
| image150x200\S001-032.jpg | 3 | yes |
| image150x200\S001-033.jpg | 4 | yes |
| image150x200\S001-034.jpg | 3 | yes |
| image150x200\S001-035.jpg | 4 | yes |
| image150x200\S001-036.jpg | 4 | yes |
| image150x200\S001-037.jpg | 2 | yes |
| image150x200\S001-038.jpg | 2 | yes |
| image150x200\S001-039.jpg | 3 | yes |
| image150x200\S001-040.jpg | 1 | yes |
| image150x200\S001-041.jpg | 1 | yes |
| image150x200\S004-001.jpg | 14 | |
| image150x200\S004-002.jpg | 11 | |
| image150x200\S004-003.jpg | 8 | |

续表

| | |
|---|---|
| image150x200\S004-004.jpg | 9 |
| image150x200\S004-005.jpg | 8 |
| image150x200\S004-006.jpg | 7 |
| image150x200\S004-007.jpg | 8 |
| image150x200\S004-008.jpg | 8 |
| image150x200\S004-009.jpg | 8 |
| image150x200\S004-010.jpg | 8 |
| image150x200\S004-011.jpg | 8 |
| image150x200\S004-012.jpg | 11 |
| image150x200\S004-013.jpg | 9 |
| image150x200\S004-014.jpg | 9 |
| image150x200\S004-015.jpg | 9 |
| image150x200\S004-016.jpg | 8 |
| image150x200\S004-017.jpg | 6 |
| image150x200\S004-018.jpg | 7 |
| image150x200\S004-019.jpg | 9 |
| image150x200\S004-020.jpg | 8 |
| image150x200\S004-021.jpg | 9 |
| image150x200\S004-022.jpg | 9 |
| image150x200\S004-023.jpg | 9 |
| image150x200\S004-024.jpg | 4　yes |
| image150x200\S004-025.jpg | 7 |
| image150x200\S004-026.jpg | 8 |
| image150x200\S004-027.jpg | 7 |
| image150x200\S004-028.jpg | 10 |

### 3. 实验分析

相似图像的相似度（汉明距离）大多在小于等于 5 的范围内。本算法中我们设置的阈值为 6，也就是说，凡是汉明距离小于 6 的，都被我们认为是相似图像，并输出“yes”作标记。本算法中查准率为 40/48×100%=83.3%，查全率为 40/42×100%=95.2%。

## 7.6 DHash 算法

DHash（Difference Hash）算法基于渐变实现，它工作在相邻像素之间。首先需要将图像缩小到 8×9（行×列）像素大小，再将图像进行灰度化。每行 9 个像素之间产生了 8 个不同的差异，一共 8 行，共产生 64 个差异值。若左边像素比右边像素更亮，则记为 1，否则记为 0。因此，我们就得到了图像对应的 64 位哈希值。本节内容将介绍 DHash 的算法实现和相关分析。

### 7.6.1 DHash 算法的使用

在 Windows7 环境下使用 Microsoft Visual Studio 2010 编译环境，并配置 OpenCV2.4.9 及以上版本。

该算法对应的项目名称为“dhashProject.sln”，位于“..\dhash”目录下。

该算法的具体使用步骤如下：

（1）打开 dhashProject 文件夹，使用 Visual Studio 2010 打开 dhashProject.sln。然后，打开项目下的源文件 dhash.cpp。

（2）输入训练数据集路径和格式替换时可在主函数下面两行代码中修改：

```
string fileFolderPath = "image150x200"; //数据集路径
string fileExtension = "jpg"; //格式
```

（3）修改输入测试数据的路径时，应在以下代码中修改：

```
sprintf(model_fileName, "image150x200\\S001-002.jpg");
```

按照以上步骤提示执行即可。

### 7.6.2 DHash 算法原理

DHash 算法是基于图像相邻像素之间的渐变。第一步，将图像尺度归一化，本算法中我们将图像缩放至 8×9（行×列）像素。当然，读者也可以根据自己的情况设定大小。第二步，将图像灰度化。第三步，计算图像的哈希值。如果左边的像素值比右边相邻的像素值高，则记为 1；反之，则记为 0。每行的 9 个像素产生

8 个差异值，一共 8 行，最后产生 64 个差异值。第四步，利用汉明距离进行相似性匹配。

其中，汉明距离是从一个哈希值到另一个哈希值需要替换（不相同）的位数。

### 7.6.3　DHash 算法实现

下面介绍一下 DHash 算法实现的具体步骤。

**1. 缩小图像**

```
resize(image1, image, Size(9,8));
```

缩小图像的尺寸为 8×9（行×列）像素。

**2. 转化成灰度图**

将第一步得到的图像转化成 256 阶的灰度图（此步骤已在读取图像时执行）。

**3. 计算哈希值**

首先 DHash 算法是基于渐变的，因此它工作在相邻像素之间。如果左边的像素值比右边相邻的像素值高，则记为 1；反之，则记为 0。每行的 9 个像素产生 8 个差异值，一共 8 行，最后产生 64 个差异值。

```
for(int i = 0 ; i < 8 ; i++)
{
    uchar* inputImageData = image.ptr<uchar>(i);
    for(int j = 0 ; j < 8 ; j++)
    {
        if(inputImageData[j]>inputImageData[j+1])
            compar.at<char>(i,j)=1;
        else
            compar.at<char>(i,j)=0;
    }
}
```

**4. 指纹（hash 值）匹配**

```
int d=0;
for(int c=0;c<64;c++)
{   if( baseDescriptor.at<float>(0,c)!=targetDescriptor.at<
```

```
float>(0,c))
            d++;
    }
```

计算两张图像哈希值之间的汉明距离（从一个哈希值到另一个哈希值需要变几位，即不同的位数）。显然，汉明距离越大，表示图片越不相似。

### 7.6.4 DHash 算法的实验数据、实验结果及分析

#### 1. 实验数据

实验的输入图像集在项目路径下，若我们想替换图像集，则在主函数的前两行进行修改。其中，第一行表示图像的路径名，第二行表示图像的类型。

输入的查询图像在.cpp 文件主函数中的代码中：

```
sprintf(model_fileName, "image150x200\\S001-002.jpg");
//查询图像
```

#### 2. 实验结果

查询图像如图 7-3 所示。



图 7-3 查询图像（2）

检索出的部分相似图像如图 7-4 所示。



图 7-4　检索出的部分相似图像（2）

图像文件名在 dhash.cpp 同一路径下的 fileName.txt 文件中，图像集中每个图像与查询图像的汉明距离保存在 dhash.cpp 同一路径下的 hanming-distance.txt 文件中，其中 fileName.txt 和 hanming-distance.txt 文件中的每一行都一一对应。部分结果如表 7-2 所示。

表 7-2　部分实验结果（2）

| | | |
|---|---|---|
| image150x200\S001-001(1).jpg | 2 | yes |
| image150x200\S001-001.jpg | 1 | yes |
| image150x200\S001-002.jpg | 0 | yes |
| image150x200\S001-003.jpg | 3 | |
| image150x200\S001-004.jpg | 3 | |
| image150x200\S001-005.jpg | 2 | yes |
| image150x200\S001-006.jpg | 3 | |
| image150x200\S001-007.jpg | 3 | |
| image150x200\S001-008.jpg | 1 | yes |
| image150x200\S001-009.jpg | 2 | yes |
| image150x200\S001-010.jpg | 1 | yes |
| image150x200\S001-011.jpg | 4 | |
| image150x200\S001-012.jpg | 3 | |
| image150x200\S001-013.jpg | 2 | yes |
| image150x200\S001-014.jpg | 2 | yes |
| image150x200\S001-015.jpg | 6 | |
| image150x200\S001-016.jpg | 6 | |
| image150x200\S001-017.jpg | 2 | yes |
| image150x200\S001-018.jpg | 3 | |
| image150x200\S001-019.jpg | 2 | yes |
| image150x200\S001-020.jpg | 1 | yes |
| image150x200\S001-021.jpg | 2 | yes |

续表

| | | |
|---|---|---|
| image150x200\S001-022.jpg | 1 | yes |
| image150x200\S001-023.jpg | 1 | yes |
| image150x200\S001-024.jpg | 1 | yes |
| image150x200\S001-025.jpg | 1 | yes |
| image150x200\S001-026.jpg | 2 | yes |
| image150x200\S001-027.jpg | 1 | yes |
| image150x200\S001-028.jpg | 2 | yes |
| image150x200\S001-029.jpg | 2 | yes |
| image150x200\S001-030.jpg | 2 | yes |
| image150x200\S001-031.jpg | 3 | |
| image150x200\S001-032.jpg | 4 | |
| image150x200\S001-033.jpg | 1 | yes |
| image150x200\S001-034.jpg | 1 | yes |
| image150x200\S001-035.jpg | 4 | |
| image150x200\S001-036.jpg | 3 | |
| image150x200\S001-037.jpg | 1 | yes |
| image150x200\S001-038.jpg | 1 | yes |
| image150x200\S001-039.jpg | 3 | |
| image150x200\S001-040.jpg | 2 | yes |
| image150x200\S001-041.jpg | 2 | yes |
| image150x200\S004-001.jpg | 8 | |
| image150x200\S004-002.jpg | 5 | |
| image150x200\S004-003.jpg | 8 | |
| image150x200\S004-004.jpg | 9 | |
| image150x200\S004-005.jpg | 7 | |
| image150x200\S004-006.jpg | 7 | |
| image150x200\S004-007.jpg | 7 | |
| image150x200\S004-008.jpg | 6 | |
| image150x200\S004-009.jpg | 8 | |
| image150x200\S004-010.jpg | 5 | |
| image150x200\S004-011.jpg | 6 | |
| image150x200\S004-012.jpg | 3 | |
| image150x200\S004-013.jpg | 4 | |
| image150x200\S004-014.jpg | 5 | |
| image150x200\S004-015.jpg | 5 | |
| image150x200\S004-016.jpg | 6 | |
| image150x200\S004-017.jpg | 5 | |

续表

| | |
|---|---|
| image150x200\S004-018.jpg | 7 |
| image150x200\S004-019.jpg | 9 |
| image150x200\S004-020.jpg | 8 |
| image150x200\S004-021.jpg | 8 |
| image150x200\S004-022.jpg | 6 |
| image150x200\S004-023.jpg | 6 |
| image150x200\S004-024.jpg | 5 |
| image150x200\S004-025.jpg | 5 |
| image150x200\S004-026.jpg | 3 |
| image150x200\S004-027.jpg | 6 |
| image150x200\S004-028.jpg | 6 |
| image150x200\S004-029.jpg | 6 |
| image150x200\S004-030.jpg | 4 |

3．实验分析

算法中我们设置的汉明距离的阈值为 3。所得到的实验结果，查全率为 28/42×100%=67.7%，但是噪声极大，查准率很低。该算法更适用于图像集中是否存在完全相同的图像的查询（而非近似图像的查询）。

## 7.7　PCA 算法

PCA（Principal Component Analysis）是一种常用的数据分析方法。它利用线性代数中的线性变换将原始图像变换成一组各维度线性无关的表示，可用于提取数据的主要特征分量，也适用于图像数据（高维数据）的降维。

在本节中，我们先将图像调整到合适的尺寸，然后再对所有图像进行 PCA 处理。这样就解决了图像间检索速度慢的问题。我们使用的 PCA 算法是非监督式的，它能很好地挑选代表所有样本的属性。下面我们来具体阐述 PCA 算法的使用与具体实现。

### 7.7.1　PCA 算法的使用

使用 Windows 7 操作系统，以及 CodeBlocks+OpenCV2.4.9 的软件环境。

*运行相关程序的步骤如下：*

（1）使用 CodeBlocks 打开“PCA+Cosdistance”项目下的 FaceResearch.cbp 文件。

（2）打开 Sources 源文件中的 main.cpp 文件，修改图像集所属文件夹名称及图像的格式。代码如下所示：

```
imgData = readImg("image30x40","jpg"); //读入所有训练集
```

（3）读入测试图像的完整路径。代码如下所示：

```
Mat singleImg = imread(filename,0); //读入测试图像
```

（4）运行 Sources 源文件中的 main.cpp 文件即可。

## 7.7.2 PCA 算法原理

对于大量多维数据来说，变量之间相互关联，多重共线性会导致解空间的不稳定，从而导致结果的不连贯；多维度数据具有稀疏性，过多的变量会阻碍我们查找数据中的内在规律。因此，为了更好、更快地计算两个图像之间的差异，我们需要对原始图像进行 PCA 降维。其作用有三：（1）减少预测变量的个数；（2）确保这些变量之间是相互独立的，去除了噪声，发现数据中的模式，从而增加变量的代表性和变量间的区分度；（3）提供一个框架来解释结果[4]。

PCA 的算法思想如下[5]：假设有 $m$ 个实例，$n$ 维数据。第一步，将原始数据按列组成 $n$ 行 $m$ 列矩阵 $\boldsymbol{X}$；第二步，将 $\boldsymbol{X}$ 的每一行进行零均值化，也就是减去这一行的均值；第三步，求出协方差矩阵，以及其特征值及对应的特征向量；第四步，将特征向量按照对应特征值的大小从上到下按行排列成矩阵，取前 $k$ 行组成矩阵 $\boldsymbol{p}$；第五步，$y=\boldsymbol{pX}$ 即为降维到 $k$ 维后的数据。PCA 的实质就是在尽可能好地代表原来特征的情况下，将原来特征进行变换、映射至低纬度空间中。

降维的实现[6]：PCA 降维的目的就是“降噪”和“去冗余”。降噪的目的就是使保留下来的维度间的相关性尽可能小，也就是说让协方差矩阵中非对角线元素都基本为零。这一步在线性代数中称为矩阵对角化。对角化后的矩阵的对角线上是协方差矩阵的特征值，它有两层含义：第一层，它是各个维度上的新方差；第二层，它是各个维度本身应该拥有的能量。这就是我们为何一直将方差与能量等同的原因。而“去冗余”的目的就是使保留下来的维度含有的“能量”（方差）尽

可能最大。首先我们需要知道，协方差矩阵度量的是维度与维度之间的关系，而非样本与样本之间。协方差矩阵的主对角线上的元素是各个维度上的方差(能量)。其他元素是两两维度间的协方差（相关性）。在除噪对角化后的协方差矩阵的对角线上较小的新方差对应的就是那些该舍掉的维度。因此，我们只取那些含有较大能量（特征值）的维度，其余的舍掉即可。PCA 线性数学原理实质其实就是对角化协方差矩阵。

PCA 的过程如下：

（1）特征中心化，即每一维的数据都减去该维的均值。这里的“维”指的就是一个特征（或属性），变换之后每一维的均值都变成了 0。假如原始数据是一个矩阵 ***A***，我们将每一列减去该列均值后，得到矩阵 ***B***。

（2）计算矩阵 ***B*** 的协方差矩阵 ***C***。

（3）计算协方差矩阵 ***C*** 的特征值和特征向量。

（4）挑选出大的特征值对应的特征向量，得到新的数据集。

### 7.7.3　PCA 算法实现

PCA 算法的具体实现步骤如下。

**1. 前期处理**

```
Mat imgData;
imgData = readImg("image30x40","jpg"); //读入所有训练集，并转换为 float
```

其中，readImg()函数如下：把灰度图像调整成大小为 30×40 像素的图像，并对图像进行 reshape、pushback 处理，存储到一个 Mat 变量里面。

```
Mat readImg(string fileFolderPath,string fileExtension)
{
 char fileName[2000];
 //to store the current input image
 Mat inputimg;
 Mat imgSet;
 struct _finddata_t fileInfo;
 string fileFolder = fileFolderPath + "/*." + fileExtension;
 long findResult = _findfirst(fileFolder.c_str(), &fileInfo);
```

```
    if (findResult == -1)
    {
      _findclose(findResult);
      return imgSet;
    }

    do
    {
     sprintf(fileName, "%s\\%s", fileFolderPath.c_str(), fileInfo.name);
     if ( fileInfo.attrib == _A_ARCH)  //是存档类型文件
     {
        cout<<fileName<<endl;
        inputimg = imread(fileName,0); //Load as grayscale
      if(inputimg.empty())
     {
        cout<<"No image"<<endl;
        return imgSet;
     }
     Mat temp = inputimg.reshape(0,1);
     imgSet.push_back(temp);
    }
  }while (!_findnext(findResult, &fileInfo));

Mat_<float> b;
imgSet.convertTo(b, CV_32FC1);
FileStorage fs("imgSet.yml", FileStorage::WRITE);
fs<<"imgSet"<<b;
fs.release();
return b;
}
```

## 2. 在图像集上进行 PCA 操作

```
PCA pca(imgData, cv::Mat(), CV_PCA_DATA_AS_ROW,0.9); //构建 PCA
Mat pcaFeature = pca.project(imgData);  //投影集合到特征空间
```

构建 PCA，其中第一个参数表示将矩阵形式表示的图像数据以行向量形式表

示，如尺寸为 30×40 像素的图像，表示为 1 行×1200 列。若有 20 张图像，此时的 imgData 就是一个 20 行×1200 列的 Mat。第三个参数 CV_PCA_DATA_AS_ROW 表示的是矩阵的每一行代表一个样本。第四个参数 0.9 表示选择构成 90%的能量的特征值，调用 OpenCV 中 PCA 算法，将 Mat imgData 投影到特征空间生成 pcaFeature。上面已经提到，PCA 降维的目的是去除噪声，发现数据中的模式；确保变量相互独立；提高运行速度。

**3．输入测试图像**

输入测试图像，展开成 1 行×1200 列，存储于一个 test-Mat 变量里，并将测试图像 test-Mat 也投影到特征空间（测试图像要经过上述两个步骤相同的处理）。

**4．求余弦相似度**

将两个图像的表示分别视为两个行向量，求这两个向量之间的内积（或称为点积）。它们之间的相似值在 0 和 1 之间浮动，1 表示和测试图像相同，余弦相似值越大，表示两个图像的相似度越大。在本实验中，相似图像（和输入图像为同一个人的图像）的值大多都在 0.90 以上。

calculateCosEuDis()函数参考本章 7.2 节相关内容。

### 7.7.4　PCA 算法的实验数据、实验结果及分析

**1．实验数据**

图像库中的图像为“..\PCA+Cosdistance\image30×40”路径下的所有（497 张）图像。

输入的查询图像为“..\PCA+Cosdistance\image30×40”路径下的 S001-001.jpg 图像。

image30×40 图像集的说明：以其中一张图像的名称为例，“s001-001.jpg”就代表第一个人的第一张图像，“s001-002.jpg”就表示第一个人的第二张图像。类似地，“s002-001.jpg”代表第二个人的第一张图像。

**2．实验结果**

以下是部分测试结果。

查询图像如图 7-5 所示。

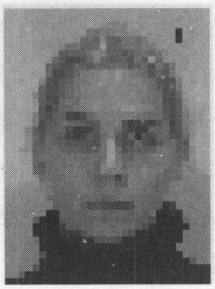


图 7-5　查询图像（3）

检索出的部分相似图像如图 7-6 所示。

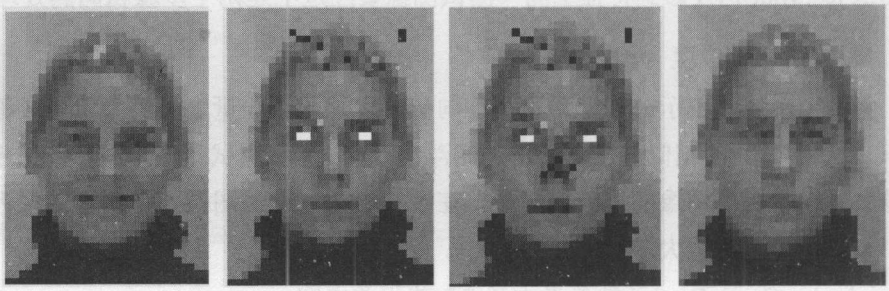


图 7-6　检索出的部分相似图像（3）

实验结果为“..\PCA+Cosdistance”路径下的 similityPCAFeature.txt 文件。其中每一行代表图像的序号和相似度。图像的顺序包含在“..\PCA+Cosdistance”路径下的 FileName.txt 文件。

表 7-3 所示是我们整理出来的部分实验结果。

**表 7-3　部分实验结果（3）**

| | | |
|---|---|---|
| image30x40\S001-001 (2).jpg | 0.925341 | yes |
| image30x40\S001-001(1).jpg | 0.917406 | yes |
| image30x40\S001-001.jpg | 0.923537 | yes |
| image30x40\S001-002.jpg | 0.93925 | yes |
| image30x40\S001-003.jpg | 0.898768 | yes |
| image30x40\S001-004.jpg | 0.947097 | yes |
| image30x40\S001-005.jpg | 0.95262 | yes |
| image30x40\S001-006.jpg | 0.954788 | yes |
| image30x40\S001-007.jpg | 0.950446 | yes |
| image30x40\S001-008.jpg | 0.947844 | yes |
| image30x40\S001-009.jpg | 0.956001 | yes |

续表

| | | |
|---|---|---|
| image30x40\S001-010.jpg | 0.913798 | yes |
| image30x40\S001-011.jpg | 0.971791 | yes |
| image30x40\S001-012.jpg | 1 | yes |
| image30x40\S001-013.jpg | 0.950791 | yes |
| image30x40\S001-014.jpg | 0.922575 | yes |
| image30x40\S001-015.jpg | 0.924854 | yes |
| image30x40\S001-016.jpg | 0.924167 | yes |
| image30x40\S001-017.jpg | 0.957395 | yes |
| image30x40\S001-018.jpg | 0.916604 | yes |
| image30x40\S001-019.jpg | 0.930222 | yes |
| image30x40\S001-020.jpg | 0.925173 | yes |
| image30x40\S001-021.jpg | 0.965218 | yes |
| image30x40\S001-022.jpg | 0.938787 | yes |
| image30x40\S001-023.jpg | 0.929929 | yes |
| image30x40\S001-024.jpg | 0.929003 | yes |
| image30x40\S001-025.jpg | 0.94608 | yes |
| image30x40\S001-026.jpg | 0.948211 | yes |
| image30x40\S001-027.jpg | 0.967817 | yes |
| image30x40\S001-028.jpg | 0.962606 | yes |
| image30x40\S001-029.jpg | 0.958662 | yes |
| image30x40\S001-030.jpg | 0.92221 | yes |
| image30x40\S001-031.jpg | 0.957307 | yes |
| image30x40\S001-032.jpg | 0.957092 | yes |
| image30x40\S001-033.jpg | 0.941929 | yes |
| image30x40\S001-034.jpg | 0.941695 | yes |
| image30x40\S001-035.jpg | 0.941668 | yes |
| image30x40\S001-036.jpg | 0.934702 | yes |
| image30x40\S001-037.jpg | 0.938823 | yes |
| image30x40\S001-038.jpg | 0.92949 | yes |
| image30x40\S001-039.jpg | 0.92825 | yes |
| image30x40\S001-040.jpg | 0.907811 | yes |
| image30x40\S001-041.jpg | 0.89276 | yes |
| image30x40\S004-001.jpg | 0.736853 | |
| image30x40\S004-002.jpg | 0.66891 | |
| image30x40\S004-003.jpg | 0.569547 | |
| image30x40\S004-004.jpg | 0.576089 | |
| image30x40\S004-005.jpg | 0.598756 | |

续表

| | |
|---|---|
| image30x40\S004-006.jpg | 0.590958 |
| image30x40\S004-007.jpg | 0.589472 |
| image30x40\S004-008.jpg | 0.626952 |
| image30x40\S004-009.jpg | 0.616425 |
| image30x40\S004-010.jpg | 0.664999 |
| image30x40\S004-011.jpg | 0.614756 |
| image30x40\S004-012.jpg | 0.698028 |
| image30x40\S004-013.jpg | 0.721973 |
| image30x40\S004-014.jpg | 0.663512 |
| image30x40\S004-015.jpg | 0.660757 |
| image30x40\S004-016.jpg | 0.651322 |
| image30x40\S004-017.jpg | 0.66814 |
| image30x40\S004-018.jpg | 0.71688 |
| image30x40\S004-019.jpg | 0.632139 |
| image30x40\S004-020.jpg | 0.611621 |
| image30x40\S004-021.jpg | 0.653381 |
| image30x40\S004-022.jpg | 0.686985 |
| image30x40\S004-023.jpg | 0.647428 |
| image30x40\S004-024.jpg | 0.716877 |
| image30x40\S004-025.jpg | 0.705268 |
| image30x40\S004-026.jpg | 0.694846 |
| image30x40\S004-027.jpg | 0.776871 |
| image30x40\S004-028.jpg | 0.77372 |
| image30x40\S004-029.jpg | 0.692416 |
| image30x40\S004-030.jpg | 0.684214 |
| image30x40\S004-031.jpg | 0.68754 |
| image30x40\S004-032.jpg | 0.698179 |
| image30x40\S004-033.jpg | 0.659384 |
| image30x40\S004-034.jpg | 0.643355 |
| image30x40\S004-035.jpg | 0.705065 |
| image30x40\S004-036.jpg | 0.694136 |
| image30x40\S004-037.jpg | 0.537075 |
| image30x40\S004-038.jpg | 0.535131 |
| image30x40\S004-039.jpg | 0.598487 |
| image30x40\S004-040.jpg | 0.699427 |
| image30x40\S004-041.jpg | 0.592751 |
| image30x40\S004-042.jpg | 0.601906 |

| | |
|---|---|
| image30x40\S004-043.jpg | 0.626459 |
| image30x40\S004-044.jpg | 0.618229 |
| image30x40\S004-045.jpg | 0.671692 |
| image30x40\S004-046.jpg | 0.674359 |
| image30x40\S004-047.jpg | 0.633052 |
| image30x40\S004-048.jpg | 0.634614 |
| image30x40\S004-049.jpg | 0.650036 |
| image30x40\S004-050.jpg | 0.683783 |
| image30x40\S004-051.jpg | 0.693137 |
| image30x40\S004-052.jpg | 0.720088 |
| image30x40\S004-053.jpg | 0.739576 |
| image30x40\S004-054.jpg | 0.757482 |
| image30x40\S004-055.jpg | 0.608908 |
| image30x40\S004-056.jpg | 0.648303 |
| image30x40\S004-057.jpg | 0.479778 |

### 3. 实验分析

本算法可用于相同或相似图像的查找。本算法中相似图像与查询图像之间的最小余弦相似度的阈值为 0.89。查准率为 43/46×100%=93.5%，查全率为 43/43×100%=100%。

## 7.8 BoF 特征

Scale Invariant Feature Transform（以下简称 SIFT）算法由 D.G.Lowe 1999 年提出，2004 年完善总结[7]。SIFT 算法是一种提取图像局部特征的算法，其对旋转、尺度、亮度变化保持不变性，对视角变化、仿射变换、噪声保持一定的稳定性。

Bag-of-Feature（以下简称 BoF）Descriptor 是用于数据分类流行的视觉描述符之一。BoF 的灵感源于一个用来进行文档分类的 Bag of Word（以下简称 BoW）。在计算机视觉中，一个 BoW 代表词汇的出现次数的稀疏向量。也就是说，一个特征视觉上的 BoW 就相当于局部图像特征的出现次数的稀疏向量。

在使用 BoF 之前，需要做图像的分解。图像的分解被描述为：利用低级特征来描述图像，如颜色、形状和纹理特征。颜色直方图、形状轮廓、边缘直方图描述符都是很流行的基于图像内容检索的技术。为了获得一个 BoF 描述符，本节中我

们使用 SIFT 算法对图像提取特征。SIFT 算法是特征提取和描述中很受欢迎的算法之一。SIFT 特征点很可能是图像中的一个角点、边缘点、暗区的亮点及亮区的暗点等。

下面是我们基于 SIFT 提取特征的 BoF 思想的算法[8]。

### 7.8.1 BoF-SIFT 算法的使用

在 Windows 7 环境下，使用 Visual Studio 2010 开发环境，配置 OpenCV2.4.9。解决方案配置选择“release”，该算法对应的项目文件名称是“BoFSIFT.sln”，位于“..\BoF-SIFT\BoFSIFT”路径下。

使用 Visual Studio 2010 打开“..\BoFSIFT\BoFSIFT”路径下的 BoFSIFT.sln 文件，运行源文件中的 BoFSIFT.cpp 文件即可。

### 7.8.2 BoF-SIFT 算法原理

首先我们了解一下 SIFT 算法的原理。SIFT 是一种检测图像局部特征的算法，该算法通过在尺度空间寻找极值点，提取位置、尺度、旋转不变量，并求特征点的描述子，从而为相似性检索提供依据。

BoF 主要分为两大步。第一步是得到 BoF 集合，可以得到特定特征包的集合，然后使用它们生成 BoF 描述符；第二步是对已给的特征聚集，然后将这些包视为一个直方图中的柱子（Bar），最后使用该直方图对图像进行分类。

### 7.8.3 BoF-SIFT 算法实现

SIFT 算法的步骤如下。

（1）构建尺度空间。

（2）检测 DoG 尺度空间极值点：为了寻找尺度空间的极值点，每一个采样点要与其所有的相邻点比较，看它是否比其图像域和尺度域的相邻点大或者小。中间的检测点和它同尺度的 8 个相邻点和上下相邻尺度对应的 9×2 个点（共 26 个点）比较，以确保在尺度空间和二维图像空间都检测到极值点。一个点如果在 DoG 尺度空间本层以及上下两层的 26 个领域中是最大或最小值，就认为该点是图像在该尺度下的一个特征点。

（3）除去不好的特征点：去掉 DoG 局部曲率不对称的像素。

（4）给每一个特征点赋值一个 128 维方向参数：上一步中确定了每张图像中的特征点，为每个特征点计算一个方向，依照此方向做下一步的计算，利用关键点邻域像素的梯度方向分布特性为每个特征点制定方向参数，使算子具备旋转不变性。其中每一个关键点有三个信息：位置、所处尺度、方向。梯度直方图的范围是 0°～360°，其中每 10°一个柱，共 36 个柱。距中心点越远的领域，其对直方图的贡献也相应越小。

在实际计算时，我们在以关键点为中心的邻域窗口内采样，并用直方图统计邻域像素的梯度方向。梯度直方图的范围是 0°～360°，其中每 45°一个柱，共 8 个柱；或者每 10°一个柱，共 36 个柱。直方图的峰值则代表了该关键点处邻域梯度的主方向，即作为该关键点的方向。

（5）关键点描述子的生成：首先将坐标轴旋转为关键点的方向，以确保旋转不变形。以关键点为中心取 16×16 像素的窗口，然后将该窗口分为 4×4=16 个子窗口。对于每个子窗口（其实就是 4×4 个像素点的区域），我们用 8 个方向来描述它，最终就得到了 16×8=128 维的描述子，每一维都可以表示 4×4 个格子中的 scale/orientation，将这个向量归一化后就进一步去除了光照的影响。需要注意的是，128 维表示的是一个描述子，也就是一个特征点，而非表示整个图像。也就是说，一个图像是由若干个 128 维的描述子组成。如若判断两个图像是否相似，现假设图像 A 是由 $m$ 个 128 维的描述子组成，图像 B 是由 $n$ 个 128 维的描述子组成，使用图像 A 的第一个特征描述子与图像 B 的所有描述子进行匹配，以此类推，直至遍历完图像 A 的所有描述子，匹配成功的个数达到预设定的阈值，我们就认为这两个图像是相似图像。但是本节算法中我们并没有采用该方法，而是采用和 BoF 思想相结合的相似性图像检索。下面我们来看一下 BoF-SIFT 结合后的算法实现。

文献[8]实现了 BoF-SIFT 算法，相关的算法步骤如下。

（1）计算图像库中的特征包集合。

①挑选出一个大的图像库。

②提取图像库中所有图像的 SIFT 特征点，得到每个特征点的 SIFT 描述符。

③使用 K-means 对所有的特征描述符集合进行聚集。

④得到一个视觉的词汇表。

（2）得到查询图像的 BoF 描述符。

①提取查询图像的 SIFT 特征点。

②得到每个特征点的 SIFT 描述符。

③与第一步生成的词汇表和特征描述符匹配。

④建立直方图。

（3）计算欧氏距离，输出查询图像与图像库中所有图像的相似度。其中，文献[8]中给出了 BoF-SIFT 算法的核心代码。

obtain_dictionary()函数如下：

```
bool obtain_dictionary()
{
    char fileName[2000];//to store the input file names;
    Mat inputimg;//to store the current input image
    vector<KeyPoint> keypoints;//To store the keypoints that will
be extracted by SIFT
    Mat descriptor;//To store the SIFT descriptor of current image
    Mat featuresUnclustered; //To store all the descriptors
extracted from all the images.
    SiftDescriptorExtractor detector;//The SIFT feature extractor
and descriptorf
    string fileFolderPath = "C:\\Users\\ZDD\\Desktop\\image";
    string fileExtension = "jpg";
    string fileFolder = fileFolderPath + "/*." + fileExtension;
    struct _finddata_t fileInfo;    //文件信息结构体
    long findResult = _findfirst(fileFolder.c_str(), &fileInfo);
//第一次查找
    if (findResult == -1)
    {
        findclose(findResult);
        return false;
    }
    do
    {
        sprintf(fileName, "%s\\%s", fileFolderPath.c_str(),
fileInfo.name);
```

```
        if ( fileInfo.attrib == _A_ARCH)  //是存档类型文件
        {
          cout<<fileName<<" ";
          inputimg = imread(fileName,0); //Load as grayscale
          if(inputimg.empty())
          {
             cout<<"No image"<<endl;
             //return 0;
          }
          //open the file
          detector.detect(inputimg, keypoints);
          //compute the descriptors for each keypoint
          detector.compute(inputimg, keypoints,descriptor);
          featuresUnclustered.push_back(descriptor);
        //put the all feature descriptors in a single Mat object
        cout<<"descriptor:"<<descriptor.rows<<"
"<<descriptor.cols<<endl;
         //rows are the number of keypoint,cols are the dimension
of every descriptor.
       }
      }
       while (!_findnext(findResult, &fileInfo));
      //Construct BOWKMeansTrainer
      //the number of bags
       int dictionarySize=300;
      //define Term Criteria
      TermCriteria tc(CV_TERMCRIT_ITER,100,0.001);
     //retries number
      int retries=1;
      //necessary flags
      int flags=KMEANS_PP_CENTERS;
      //Create the BoW (or BoF) trainer
      BOWKMeansTrainer bowTrainer(dictionarySize,tc,retries,flags);
      //cluster the feature vectors
      Mat dictionary=bowTrainer.cluster(featuresUnclustered);
      //store the vocabulary
```

```
    FileStorage fs("dictionary.yml", FileStorage::WRITE);
    fs << "vocabulary" << dictionary;
    fs.release();
    return true;
}
```

obtain_BoF(char * )函数如下[8]：

```
Mat obtain_BoF(char * filename)
{
 //prepare BOW descriptor extractor from the dictionary
    Mat dictionary;
    FileStorage fs("dictionary.yml", FileStorage::READ);
    fs["vocabulary"] >> dictionary;
    fs.release();

    //create a nearest neighbor matcher
    Ptr<DescriptorMatcher> matcher(new FlannBasedMatcher);
    //create Sift feature point extracter
    Ptr<FeatureDetector> detector(new SiftFeatureDetector());
    //create Sift descriptor extractor
    Ptr<DescriptorExtractor> extractor(new SiftDescriptorExtractor);
    //create BoF (or BoW) descriptor extractor
    BOWImgDescriptorExtractor bowDE(extractor,matcher);
    //Set the dictionary with the vocabulary we created in the first step
    bowDE.setVocabulary(dictionary);

    //To store the image file name
    //char * filename = new char[100];

    //To store the image tag name - only for save the descriptor in
a file
    char * imageTag = new char[10];

    //open the file to write the resultant descriptor
    FileStorage fs1("descriptor.yml", FileStorage::WRITE);
```

```
    //the image file with the location. change it according to your
image file location
    //sprintf(filename,"image1/S001-001.jpg");
    //read the image
    Mat img=imread(filename,CV_LOAD_IMAGE_GRAYSCALE);
    //To store the keypoints that will be extracted by SIFT
    vector<KeyPoint> keypoints;
    //Detect SIFT keypoints (or feature points)
    detector->detect(img,keypoints);
    //To store the BoW (or BoF) representation of the image
    Mat bowDescriptor;
    //extract BoW (or BoF) descriptor from given image
    bowDE.compute(img,keypoints,bowDescriptor);

    FILE * pfile = fopen("myFileS001-002.csv","a");
    for(int c=0; c<bowDescriptor.cols; c++)
    {
        if(c!=(bowDescriptor.cols-1))
            fprintf(pfile,"%f,",bowDescriptor.at<float>(0,c));
        else
            fprintf(pfile,"%f\n",bowDescriptor.at<float>(0,c));
    }
    //prepare the yml (some what similar to xml) file
    sprintf(imageTag,"img1");
    //write the new BoF descriptor to the file
    fs1 << imageTag << bowDescriptor;
    //release the file storage
    fs1.release();
    return bowDescriptor;
}
```

计算查询图像与图像库中的每个图像的欧氏距离（此算法中 SIFT 提取出来特征为 128 维度）。calculateEuDis()函数参考本书 7.2 节。

### 7.8.4 BoF-SIFT 算法的实验数据、实验结果及分析

#### 1. 实验数据

输入图像为“..\BoFSIFT\image”路径下的所有（495 幅）图像。

输入测试图像为“..\BoFSIFT\image\S001-008.jpg”图像。

image 图像集的说明：以其中一幅图像的名称为例，“s001-001.jpg”就代表第一个人的第一幅图像，“s001-002.jpg”就表示第一个人的第二幅图像，“s002-001.jpg”代表第二个人的第一幅图像。

#### 2. 实验结果

实验结果存储在“..\BoFSIFT\BoFSIFT\simility.txt”。以下是欧氏距离度量方法的部分查询结果。

查询图像如图 7-7 所示。



图 7-7 查询图像（4）

检索出的部分相似图像如图 7-8 所示。



图 7-8 检索出的部分相似图像（4）

表 7-4 所示是利用欧氏距离作为度量依据，利用 naïve 查询处理算法来遍历图像库中的每幅图像求得的结果。

表 7-4　实验结果

| | |
|---|---|
| image\S001-001.jpg 0.0455566 | image\S004-009.jpg 0.0795901 |
| image\S001-002.jpg 0.0468523 | image\S004-010.jpg 0.0992301 |
| image\S001-003.jpg 0.166601 | image\S004-011.jpg 0.0791963 |
| image\S001-004.jpg 0.036611 | image\S004-012.jpg 0.0851957 |
| image\S001-005.jpg 0.0384667 | image\S004-013.jpg 0.081527 |
| image\S001-006.jpg 0.0393293 | image\S004-014.jpg 0.0834162 |
| image\S001-007.jpg 0.0436583 | image\S004-015.jpg 0.0844262 |
| image\S001-008.jpg 0.0121794 | image\S004-016.jpg 0.0812988 |
| image\S001-009.jpg 0.0399772 | image\S004-017.jpg 0.0840907 |
| image\S001-010.jpg 0.0382122 | image\S004-018.jpg 0.0799451 |
| image\S001-011.jpg 0.0433705 | image\S004-019.jpg 0.0818624 |
| image\S001-012.jpg 0.0424281 | image\S004-020.jpg 0.0829646 |
| image\S001-013.jpg 0.0488037 | image\S004-021.jpg 0.0854618 |
| image\S001-014.jpg 0.106697 | image\S004-022.jpg 0.0841607 |
| image\S001-015.jpg 0.0651336 | image\S004-023.jpg 0.0825589 |
| image\S001-016.jpg 0.0425317 | image\S004-024.jpg 0.0850435 |
| image\S001-017.jpg 0.0427735 | image\S004-025.jpg 0.084033 |
| image\S001-018.jpg 0.0449644 | image\S004-026.jpg 0.0817309 |
| image\S001-019.jpg 0.0466363 | image\S004-027.jpg 0.0829875 |
| image\S001-020.jpg 0.0440568 | image\S004-028.jpg 0.0814102 |
| image\S001-021.jpg 0.0409154 | image\S004-029.jpg 0.0832851 |
| image\S001-022.jpg 0.0410279 | image\S004-030.jpg 0.0817433 |
| image\S001-023.jpg 0.0423409 | image\S004-031.jpg 0.0813276 |
| image\S001-024.jpg 0.0673626 | image\S004-032.jpg 0.0812025 |
| image\S001-025.jpg 0.042353 | image\S004-033.jpg 0.0793984 |
| image\S001-026.jpg 0.0382079 | image\S004-034.jpg 0.0840991 |
| image\S001-027.jpg 0.0420996 | image\S004-035.jpg 0.0856628 |
| image\S001-028.jpg 0.0405814 | image\S004-036.jpg 0.0820133 |
| image\S001-029.jpg 0.0413214 | image\S004-037.jpg 0.0838138 |
| image\S001-030.jpg 0.044649 | image\S004-038.jpg 0.0837384 |
| image\S001-031.jpg 0.0486594 | image\S004-039.jpg 0.0813016 |
| image\S001-032.jpg 0.0477016 | image\S004-040.jpg 0.116886 |
| image\S001-033.jpg 0.0467774 | image\S004-041.jpg 0.0786953 |

续表

| | |
|---|---|
| image\S001-034.jpg 0.0438553 | image\S004-042.jpg 0.0819111 |
| image\S001-035.jpg 0.0429644 | image\S004-043.jpg 0.0802983 |
| image\S001-036.jpg 0.0517051 | image\S004-044.jpg 0.0808112 |
| image\S001-037.jpg 0.063424 | image\S004-045.jpg 0.086619 |
| image\S001-038.jpg 0.0484854 | image\S004-046.jpg 0.0909819 |
| image\S001-039.jpg 0.052259 | image\S004-047.jpg 0.0804728 |
| image\S001-040.jpg 0.125454 | image\S004-048.jpg 0.0802212 |
| image\S001-041.jpg 0.0955797 | image\S004-049.jpg 0.0798425 |
| image\S004-001.jpg 0.076818 | image\S004-050.jpg 0.0805512 |
| image\S004-002.jpg 0.0805555 | image\S004-051.jpg 0.0848007 |
| image\S004-003.jpg 0.0836806 | image\S004-052.jpg 0.0811868 |
| image\S004-004.jpg 0.0821294 | image\S004-053.jpg 0.078522 |
| image\S004-005.jpg 0.0784942 | image\S004-054.jpg 0.0806904 |
| image\S004-006.jpg 0.0788286 | image\S004-055.jpg 0.0797906 |
| image\S004-007.jpg 0.077422 | image\S004-056.jpg 0.0795923 |
| image\S004-008.jpg 0.0808385 | image\S004-057.jpg 0.0793834 |
| | image\S004-058.jpg 0.077408 |
| | image\S004-059.jpg 0.0793027 |

3．实验分析

使用余弦相似度度量方法：设置余弦相似度阈值为 0.82。实验结果查准率为 29/42×100%=69.05%，查全率为 29/42×100%=69.05%。

使用欧氏距离度量方法：设置欧氏距离相似度阈值为 0.5。查准率为 30/60×100%=50%，查全率为 30/42×100%=71.43%。

由于设定的阈值不同，查全率和查准率会有所变化。

## 7.9 用于图像快速检索的 KD-Tree 索引

在图像和视频处理领域中，数据通常是高维的，此时，最相似 K 近邻的查找非常费时。图像处理领域的研究人员经常使用 KD-Tree 查找与输入图像的特征相似的图像。KD-Tree 能够大大节省 K 近邻查找的运行时间，是高维度特征相似性检索不可或缺的利器。本书中使用 FLANN 库，它是最完整的近邻查询算法的开

源库，包含 KD-Tree 的实现。在本章中，我们仅使用了 flann::Index_::radiusSearch，其根据给定的查询数据，执行基于半径的最近邻检索。

### 7.9.1　FLANN 算法的使用

该算法的运行环境是在 Linux 下的 Ubuntu 系统，使用命令行来运行。从 https://github.com/jlblancoc/nanoflann 中下载 FLANN 算法的源码[9]，按照下载文档中的 readme.txt 文档编译.cpp 文件。其中 pointcloud_kdd_radius.cpp 是关键，我们已经将所想实现功能的代码放入 pointcloud_kdd_radius.cpp 中。打开命令窗口，进入“/nanoflann-master/nanoflann-master/build/bin”路径，运行 pointcloud_kdd_ radius.cpp 文件即可。

### 7.9.2　KD-Tree 的创建与查询处理

KD-Tree 的创建：找出数据集中方差最高的维度 $k_i$，在 $k_i$ 维度上选择中值并将本维度上的数据划分为两部分，此时我们得到两个子集合，同时创建了一个树节点，接着再对两个子集重复上述过程，直到所有子集合都不能再划分为止，此时子集中的数据保存在叶子节点。

但是我们为什么会选中数据集中方差最大的维度呢？我们举个简单的例子。现在要切一根躺着的木条，按照“轮着来”的方法先竖着切一刀，木条一分为二，接下来就是横着切一刀，这时候就有点考验刀法了，如果木条的直径（横截面）较大，还可以轻松下手；如果木条的直径很小，就没有办法切了。因此，如果 $k$ 维数据的分布像一块正方体的豆腐一样，“轮着来”的切分法可能还是奏效的，但是如果 $k$ 维度的数据分布像木条一样，“轮着来”就不好用了。我们想象一下，$k$ 维数据集合的分布像木条一样，也就是说这些数据在木条较长方向上的维度比较大，在该方向维度上的方差比较大，数据分散得比较开，因此我们就更容易在这个数据维度上将它们划分开。于是，这就引出了我们选择维度的另一种方法——最大方差法，即每次我们选择方差最大的维度进行划分。

在 OpenCV 中，KeyPoint Matching 的方法有 Brute-force matcher（简单粗暴匹配），用暴力方法找到点集中每一个 descriptor（描述符）与已知点距离最近的 descriptor（描述符）；而 Flann-based matcher 使用快速近似最近邻检索方法查询。

KD-Tree 的查询基于深度检索。首先从根开始，进行深度检索并不断记录、更新当前最近节点，并将错过的节点置于优先队列中，直到遍历到叶子节点，此时再从队列中取出目前 $k_i$ 维度上距离最小的节点，然后重复上述过程，遍历至叶子节点，直到优先队列为空，或迭代次数超过 200 遍时停止。

### 7.9.3 FLANN 中 KD-Tree 的算法实现

下面给出了 FLANN 中实现的 KD-Tree 算法。确定 K 近邻的个数时，它主要依据的是查询半径参数，即 radiusSearch()函数。

```
#define DIMEN 128    //数据维度
#define NUM  497    //测试图像的个数
#define EuDistance 1   //1 表示欧氏距离;0 表示相似余弦距离
template <typename num_t>
void kdtree_demo(const size_t N)
{
    PointCloud<num_t> cloud;
    //Generate points:
    generateRandomPointCloud(cloud, N);
    //construct a kd-tree index:
    typedef KDTreeSingleIndexAdaptor<
    L2_Simple_Adaptor<num_t, PointCloud<num_t> > ,
                    PointCloud<num_t>,
                    DIMEN /* dim */
                    > my_kd_tree_t;
    my_kd_tree_t   index(DIMEN/*dim*/, cloud,
KDTreeSingleIndexAdaptorParams (10 /* max leaf */) );
    index.buildIndex();
#if 0
    //Test resize of dataset and rebuild of index:
    cloud.pts.resize(cloud.pts.size()*0.5);
    index.buildIndex();
#endif
        num_t query_pt[DIMEN];
     std::string testfilename="/host/desktop/t.txt";
     vector<double> testdata=typestr(testfilename);
```

```
    cout<<"%%%%%%%%%%"<<endl;
        for(size_t i =0;i<testdata.size();i++){
            query_pt[i] = testdata[i];
        }
    /*for(int i=0;i<DIMEN;i++)
    {
    cout<<query_pt[i]<<" ";
    }*/
        // ----------------------------------------------------------------
        // radiusSearch():  Perform a search for the N closest points
        // ----------------------------------------------------------------
        {
            const num_t search_radius = static_cast<num_t>(77.6);
            std::vector<std::pair<size_t,num_t> >   ret_matches;

            nanoflann::SearchParams params;
            //params.sorted = false;
            const size_t nMatches = index.radiusSearch(&query_pt[0],
search_radius, ret_matches, params);

            string path = "/host/desktop/filelist.txt";
            string* fileList=new string[NUM];
            readImgNameList(path,fileList);
            cout << "radiusSearch(): radius=" << search_radius << " ->
" << nMatches << " matches\n";
           for (size_t i=0; i<nMatches; i++)
            {
            cout << "idx["<< i << "]=" << ret_matches[i].first <<"
imgName is: "<<fileList[ret_matches[i].first]<< "   dist["<< i << "]="
<< ret_matches[i].second <<endl;
            }
    }
    }
```

### 7.9.4 FLANN 算法的实验数据、实验结果及分析

image 图像集的说明：以其中一些图像的名称为例，“s001-001.jpg”就代表第一个人的第一幅图像，“s001-002.jpg”就表示第一个人的第二幅图像，“s002-001.jpg”代表第二个人的第一幅图像，以此类推。

**1. 实验数据**

图像集 image 文件中的 497 幅图像经过深度学习提取得到的图像特征（保存在.txt 文件中，其中每一行代表一幅图像的特征）。

**2. 实验结果**

下面的图片是我们所做实验的截图（见图 7-9、图 7-10）和代码运行结果显示（见图 7-11）。其中运行结果显示中，每一行的三个参数分别代表索引号、图像文件名、与检索图像之间的距离。

查询图像如图 7-9 所示。



图 7-9 查询图像（5）

检索出的部分相似图像如图 7-10 所示。



图 7-10 检索出的部分相似图像（5）

```
radiusSearch(): radius=77.6 -> 38 matches
idx[0]=1    imgName is: /image/S001-001(1).jpg    dist[0]=0
idx[1]=2    imgName is: /image/S001-001.jpg    dist[1]=32.1253
idx[2]=9    imgName is: /image/S001-008.jpg    dist[2]=35.4252
idx[3]=8    imgName is: /image/S001-007.jpg    dist[3]=36.4303
idx[4]=5    imgName is: /image/S001-004.jpg    dist[4]=39.8094
idx[5]=30    imgName is: /image/S001-029.jpg    dist[5]=39.8428
idx[6]=6    imgName is: /image/S001-005.jpg    dist[6]=40.0923
idx[7]=27    imgName is: /image/S001-026.jpg    dist[7]=46.2906
idx[8]=3    imgName is: /image/S001-002.jpg    dist[8]=47.1549
idx[9]=7    imgName is: /image/S001-006.jpg    dist[9]=47.176
idx[10]=33    imgName is: /image/S001-032.jpg    dist[10]=48.0715
idx[11]=4    imgName is: /image/S001-003.jpg    dist[11]=49.4287
idx[12]=13    imgName is: /image/S001-012.jpg    dist[12]=50.331
idx[13]=41    imgName is: /image/S001-040.jpg    dist[13]=52.3365
idx[14]=12    imgName is: /image/S001-011.jpg    dist[14]=52.63
idx[15]=39    imgName is: /image/S001-038.jpg    dist[15]=55.8965
idx[16]=16    imgName is: /image/S001-015.jpg    dist[16]=57.6894
idx[17]=31    imgName is: /image/S001-030.jpg    dist[17]=58.0703
idx[18]=35    imgName is: /image/S001-034.jpg    dist[18]=58.1103
idx[19]=26    imgName is: /image/S001-025.jpg    dist[19]=58.1131
idx[20]=15    imgName is: /image/S001-014.jpg    dist[20]=59.228
idx[21]=19    imgName is: /image/S001-018.jpg    dist[21]=59.3468
idx[22]=38    imgName is: /image/S001-037.jpg    dist[22]=59.9254
idx[23]=32    imgName is: /image/S001-031.jpg    dist[23]=60.5582
idx[24]=17    imgName is: /image/S001-016.jpg    dist[24]=61.1867
idx[25]=18    imgName is: /image/S001-017.jpg    dist[25]=61.469
idx[26]=374    imgName is: /image/S018-103.jpg    dist[26]=61.7045
idx[27]=14    imgName is: /image/S001-013.jpg    dist[27]=65.4437
idx[28]=42    imgName is: /image/S001-041.jpg    dist[28]=69.3176
idx[29]=29    imgName is: /image/S001-028.jpg    dist[29]=71.0589
idx[30]=40    imgName is: /image/S001-039.jpg    dist[30]=71.4668
idx[31]=10    imgName is: /image/S001-009.jpg    dist[31]=72.5682
idx[32]=28    imgName is: /image/S001-027.jpg    dist[32]=74.7113
idx[33]=21    imgName is: /image/S001-020.jpg    dist[33]=75.0509
idx[34]=96    imgName is: /image/S004-054.jpg    dist[34]=76.6648
idx[35]=36    imgName is: /image/S001-035.jpg    dist[35]=77.0757
idx[36]=83    imgName is: /image/S004-041.jpg    dist[36]=77.361
idx[37]=34    imgName is: /image/S001-033.jpg    dist[37]=77.56
```

图 7-11　代码运行结果显示

### 3. 实验分析

我们可以从实验结果的截图中看到，我们使用的 radius（半径）的阈值为 77.6，匹配到 38 幅图像，其中匹配到的图像中有 35 幅是我们想要查找到的，有 3 幅不是我们想要的，另外还有 7 幅图像没有匹配出来。因此，查准率为 35/38×100%=92.1%，查全率为 35/42×100%=83.3%。

## 7.10　Gabor 算法

在图像处理领域，Gabor 滤波器通常用于纹理分析和特征提取等方面，这主要是因为 Gabor 滤波器在频率和方向上的表达类似于人类视觉系统。本节我们将使用 Gabor 算法对图像进行处理，并结合余弦相似度、欧氏距离度量方法及 Naïve 检索方法对图像进行相似性检索。

### 7.10.1 Gabor 算法的使用

在 Windows 7 环境下，安装 Microsoft Visual Studio 2010 软件，并配置 OpenCV2.4.9。

本算法对应的项目名称是 texture。打开 texture 文件夹，然后使用 Visual Studio 2010 编译工具打开 texuture.sln 文件，并配置 OpenCV2.4.9 及以上版本。

数据输入可在主函数的以下两行代码中修改：

```
string fileExtension="jpg";
string fileFolderPath="D:\\desktop\\image1";//filename
```

第一行是图像的格式，第二行是图像文件夹的名称。注：目前该代码只能实现 RGB 图像。

图像的缩放比例参数在 doGabor()代码中修改：

```
double scale = 0.5;
```

结果的输出在当前项目路径下的“CosEuDis_similityGABORFeature1.txt”或“EuDis_similityGABORFeature1.txt”文件中。其中前者是利用相似余弦距离匹配后的结果，后者是利用欧氏距离匹配后的结果。注：本算法中，相似余弦距离的阈值为 0.50，欧氏距离的阈值为 62000。

测试图像为 image1 文件夹下的“\\image\\S001-002.jpg”图像。

测试图像集在当前项目下，文件名称为 image1。

按照以上步骤，执行 texture.cpp 文件即可。

### 7.10.2 Gabor 算法原理

在空间域中，二维的 Gabor 滤波器由高斯核函数和正弦平面波的乘积组成。Gabor 函数的数学公式如下[10,11]：

$$G(x,y,\lambda,\theta,\varphi,\sigma,\gamma)=\exp(-\frac{x'^2+\gamma^2y'^2}{2\sigma^2})\exp(i(2\pi\frac{x'}{\lambda}+\varphi))$$

实数部分：

$$G(x,y,\lambda,\theta,\varphi,\sigma,\gamma)=\exp(-\frac{x'^2+\gamma^2y'^2}{2\sigma^2})\cos(2\pi\frac{x'}{\lambda}+\varphi)$$

虚数部分：

$$G(x,y,\lambda,\theta,\varphi,\sigma,\gamma)=\exp(-\frac{x'^2+\gamma^2y'^2}{2\sigma^2})\sin(2\pi\frac{x'}{\lambda}+\varphi)$$

其中，$x'=x\cos\theta+y\sin\theta$, $y'=-x\sin\theta+y\cos\theta$。

$(x,y)$表示图像中像素的位置（坐标）。

$\lambda$：波长，该参数以像素为单位。

$\theta$：取值范围 0~2π，是 Gabor 函数并行条纹的方向，它决定了 Gabor 条纹的方向。

$\varphi$：指相位偏移，决定了 Gabor 函数的对称性，如 $\varphi$=0 时，为中心对称。

$\sigma$：高斯函数的标准差。

$\gamma$：空间的宽高比，它决定了 Gabor 条纹的尺度（大小）。

下面讲述 Gabor 滤波器的相关参数调优。

参考文献[10]基于 C++实现了 Gabor，运行上述程序时需要调整 Gabor 函数的参数。下面是通过调整参数得到的不同 Gabor 滤波结果，分别在图 7-12、图 7-13、图 7-14 中进行展示。

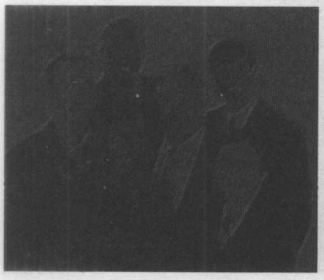
（1）调整参数 $\theta$ 后的 Gabor 滤波结果比较（见图 7-12），其中其他参数为：$\gamma$=1.0，$\lambda$=10.0，$\sigma$=100,0。



（1）原图



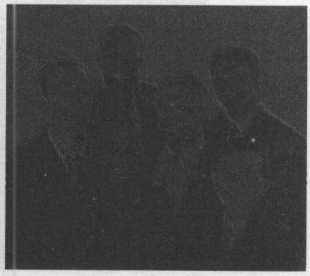
（2）$\theta$=0



（3）$\theta=\pi/4$



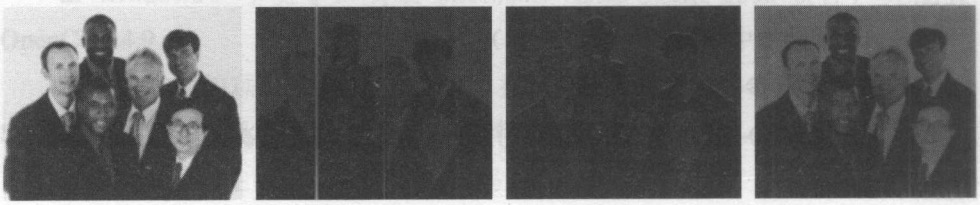
（4）$\theta=\pi/2$



（5）$\theta=3\times\pi/4$

图 7-12　调整参数 $\theta$ 后的 Gabor 滤波结果

（2）调整参数 $\lambda$ 后的 Gabor 滤波结果比较（见图 7-13），其中其他参数为：$\theta$=0，$\gamma$=1.0，$\sigma$= 100,0。



（1）$\lambda$=1.0　（2）$\lambda$=6.0　（3）$\lambda$=11　（4）$\lambda$=16

图 7-13　调整参数 $\lambda$ 后的 Gabor 滤波结果

从上面的示例可以看出，Gabor 滤波器具有方向选择性（$\theta$）和尺度选择性（$\lambda$）。此外，对滤波器输出产生影响的参数还有高斯函数的作用范围，即高斯窗体。不同高斯窗体的滤波结果如图 7-14 所示，其中 width 为滤波宽度，height 为滤波高度。



（1）width=height=5　（2）width=height=10



（3）width=height=15　（4）width=height=20

图 7-14　调整参数 width 和 height 后的 Gabor 滤波结果

### 7.10.3　Gabor 算法实现

在主函数中，读取每一幅图像，并调用 doGabor()函数获得该图像 Gabor 处理后的特征，进而做 PCA 降维处理，再进行匹配。本书使用了文献[12]中实现的 Gabor 滤波程序，相关代码如下：

```
int main(int argc, char* argv[])
{
    cv::Mat gaborSet;
    struct _finddata_t fileInfo;//文件信息结构体
    //第一次查找
    string fileExtension="jpg";
    string fileFolderPath="D:\\desktop\\image1";//filename
    string fileFolder = fileFolderPath + "\\*." + fileExtension;
    long findResult = _findfirst(fileFolder.c_str(), &fileInfo);

    if (findResult == -1)
    {
        _findclose(findResult);
        return 0;
    }
    do
    {
        char fileName[2000];
        sprintf(fileName, "%s\\%s", fileFolderPath.c_str(), fileInfo.
name);

        if ( fileInfo.attrib == _A_ARCH)  //是存档类型文件
        {
            cout<<fileName<<endl;
        }
        IplImage  * averageimage=NULL;
        averageimage=doGabor(fileName);

        cv::Mat Mataverageimage(averageimage,true);
        Mat temp=Mataverageimage.reshape(0,1);
```

```
        std::cout<<temp.rows<<"  "<<temp.cols<<endl;
        gaborSet.push_back(temp);

    }
    while (!_findnext(findResult, &fileInfo));
    std::cout<<gaborSet.rows<<"  "<<gaborSet.cols<<endl;
    /*  Mat_<float> b1;
    gaborSet.convertTo(b1, CV_32FC1);
    FileStorage fs1("gaborSet.yml", FileStorage::WRITE);
    fs1<<"gaborSet"<<b1;
    fs1.release();*/

    //PCA 提取图像特征与分析
    PCA pca1(gaborSet, cv::Mat(), CV_PCA_DATA_AS_ROW,0.9); //构建 PCA
    Mat pcaFeature1 = pca1.project(gaborSet);  //投影集合到特征空间
    cout<<"-----------------"<<endl;

    cout<<"---------2--------"<<endl;
    char* testImgName="D:\\desktop\\image\\S001-002.jpg";
    IplImage  * testimage=NULL;
    testimage=doGabor(testImgName);

    cv::Mat Mattestimage(testimage,true);
    Mat testtemp=Mattestimage.reshape(0,1);
    std::cout<<testtemp.rows<<"  "<<testtemp.cols<<endl;
    Mat testgaborSet;
    testgaborSet.push_back(testtemp);


    std::cout<<testgaborSet.rows<<"  "<<testgaborSet.cols<<endl;
    //PCA pca1(testgaborSet, cv::Mat(),CV_PCA_DATA_AS_ROW,0.9);//构建 PCA
    Mat baseDescriptor = pca1.project(testgaborSet);

    //计算相似度，以下我们尝试了两种相似性匹配
    Matcher matcher;
    cout<<"calculate simility ..."<<endl;
```

```
    ofstream outfile1("CosEuDis_similityGABORFeature1.txt");
    ofstream outfile2("EuDis_similityGABORFeature1.txt");
    for(int r=0; r<pcaFeature1.rows; r++)
    {
        Mat targetDescriptor1 = pcaFeature1.row(r);
        double similty1 = matcher.calculateCosEuDis
(targetDescriptor1, baseDescriptor);   //计算余弦相似度
        if(similty1>0.67)
            outfile1<<r<<" "<<similty1<<"            yes"<<endl;
        else
            outfile1<<r<<" "<<similty1<<endl;

        double similty2 = matcher.calculateEuDis (targetDescriptor1,
baseDescriptor);     //计算欧氏距离
        if(similty2<62000)
            outfile2<<r<<" "<<similty2<<"          yes"<<endl;
        else
            outfile2<<r<<" "<<similty2<<endl;
    }
    Sleep(10000000);
    return 1;
}
//其中 doGabor()函数的具体实现来自文献[12]
IplImage* doGabor(char* fileName)
{
    IplImage  *ipl = NULL;
    IplImage  * newip1 = NULL;
    IplImage  * newip2 = NULL;
    IplImage  * averageimage=NULL;
    ipl = cvLoadImage(fileName, CV_LOAD_IMAGE_ANYCOLOR);

    IplImage *desc;
    CvSize sz;
    double scale = 0.5;
    sz.width = ipl->width*scale;
    sz.height = ipl->height*scale;
```

```
desc = cvCreateImage(sz,ipl->depth,ipl->nChannels);
cvResize(ipl,desc,CV_INTER_CUBIC);

newip1=cvCreateImage(cvGetSize(desc),IPL_DEPTH_8U,1);
cvCvtColor (desc,newip1,CV_BGR2GRAY);
newip2 = cvCreateImage(cvGetSize(newip1), 8, 1);
averageimage = cvCreateImage(cvGetSize(newip1), 8, 1);
cvSetZero(averageimage);

for(int i=0; i<3; i++)
    for (int j=0; j<4; j++)
    {
      {
          gaborfilter(newip1,i,j,newip2);
          for(int m=0; m<newip2->height; m++)
          {
              for (int n=0; n<newip2->width; n++)
              {
                  CvScalar s1,s2,s3;
                  s1=cvGet2D(newip2, m, n);
                  s2=cvGet2D(averageimage, m, n);
                  s3.val[0]=(s1.val[0])/12+s2.val[0];
                  cvSet2D(averageimage,m,n,s3);
              }
          }
      }
    }
for (int i1=0; i1<averageimage->height; i1++)
{
    for (int j1=0; j1<averageimage->width; j1++)
    {
        CvScalar s4;
        s4=cvGet2D(averageimage, i1, j1);
        if (s4.val[0]>121)
        {
            s4.val[0]=255;
```

```
            }
            else
            {
                s4.val[0]=0;
            }
            cvSet2D(averageimage,i1,j1,s4);
        }
    }
    return averageimage;
}
```

gaborfilter()的相关具体实现如下：

```
void gaborfilter(IplImage*image1, int m,int n, IplImage* image2)
{
    int a=2,k=8,f=16;
    int Sx=2,Sy=3;
    double xPrime, yPrime, theta,t1,t2,t3;
    double aa[5][7]= {0};
    theta=(n*1.0/k)*PI;
    t3=1/(2*PI*Sx*Sy);
    for(int x=-Sx; x<=Sx; x++)
        for(int y=-Sy; y<=Sy; y++)
        {
            {
                xPrime =t3*pow(double(a),-double(m))*(x * cos(theta)
+ y * sin(theta));
                yPrime =t3*pow(double(a),-double(m))*(y * cos(theta)
- x * sin(theta));
                t1=-0.5*((xPrime/Sx)*(xPrime/Sx)+(yPrime/Sy)*(yPrime/
Sy));
                t2=2*PI*f*xPrime;
                double p1,p2,p3;
                p1=exp(t1);
                p2=cos(t2);
                p3=p1*p2;
                aa[x+2][y+3]=p3;
```

```
            }
        }
    CvMat am;
    am = cvMat(5,7,CV_64F,aa);
    cvFilter2D(image1,image2,&am,cvPoint(-1,-1));
}
```

## 7.10.4 Gabor 算法的实验数据、实验结果及分析

### 1. 实验数据

该算法的实验数据为当前路径下的 image1 文件夹下的所有 JPG 图像。原图像的尺寸为 1200×1600 像素，在本次实验中，我们设置的尺寸为 0.5，也就是说，我们将图像的尺寸缩放到 600×800 像素。

### 2. 实验结果

查询图像如图 7-15 所示。



图 7-15　查询图像（6）

检索出的部分相似图像如图 7-16 所示。



图 7-16　检索出的部分相似图像（6）

以下是实验结果的两个表格，其中，表 7-5 表示图像库中的图像与检索图像之间的余弦相似度，表 7-6 表示图像库中的图像与检索图像之间的欧氏距离。“yes”表示返回机器视觉上的相似图像，换而言之，表示在阈值范围内的相似图像。

基于余弦相似度度量的 Gabor 图像检索算法的实验结果如表 7-5 所示。

表 7-5　基于余弦相似度度量的 Gabor 图像检索算法的实验结果

| | | |
|---|---|---|
| D:\desktop\image1\S001-001.jpg | 0.274809 | |
| D:\desktop\image1\S001-002.jpg | 1 | yes |
| D:\desktop\image1\S001-003.jpg | 0.44852 | |
| D:\desktop\image1\S001-004.jpg | 0.680298 | yes |
| D:\desktop\image1\S001-005.jpg | 0.639608 | yes |
| D:\desktop\image1\S001-006.jpg | 0.679859 | yes |
| D:\desktop\image1\S001-007.jpg | 0.392323 | |
| D:\desktop\image1\S001-008.jpg | 0.585912 | yes |
| D:\desktop\image1\S001-009.jpg | 0.549033 | yes |
| D:\desktop\image1\S001-010.jpg | 0.535116 | yes |
| D:\desktop\image1\S001-011.jpg | 0.542045 | yes |
| D:\desktop\image1\S001-012.jpg | 0.592752 | yes |
| D:\desktop\image1\S001-013.jpg | 0.412401 | |
| D:\desktop\image1\S001-014.jpg | 0.427947 | |
| D:\desktop\image1\S001-015.jpg | 0.598001 | yes |
| D:\desktop\image1\S001-016.jpg | 0.6419 | yes |
| D:\desktop\image1\S001-017.jpg | 0.574348 | yes |
| D:\desktop\image1\S001-018.jpg | 0.657538 | yes |
| D:\desktop\image1\S001-019.jpg | 0.425466 | |
| D:\desktop\image1\S001-020.jpg | 0.541776 | yes |
| D:\desktop\image1\S001-021.jpg | 0.577331 | yes |
| D:\desktop\image1\S001-022.jpg | 0.409281 | |
| D:\desktop\image1\S001-023.jpg | 0.769415 | yes |
| D:\desktop\image1\S001-024.jpg | 0.672619 | yes |
| D:\desktop\image1\S001-025.jpg | 0.615536 | yes |
| D:\desktop\image1\S001-026.jpg | 0.610176 | yes |
| D:\desktop\image1\S001-027.jpg | 0.774663 | yes |
| D:\desktop\image1\S001-028.jpg | 0.66028 | yes |

续表

| | | |
|---|---|---|
| D:\desktop\image1\S001-029.jpg | 0.531597 | yes |
| D:\desktop\image1\S001-030.jpg | 0.480539 | |
| D:\desktop\image1\S001-031.jpg | 0.515336 | yes |
| D:\desktop\image1\S001-032.jpg | 0.659111 | yes |
| D:\desktop\image1\S001-033.jpg | 0.605387 | yes |
| D:\desktop\image1\S001-034.jpg | 0.633109 | yes |
| D:\desktop\image1\S001-035.jpg | 0.495985 | |
| D:\desktop\image1\S001-036.jpg | 0.502435 | yes |
| D:\desktop\image1\S001-037.jpg | 0.502083 | yes |
| D:\desktop\image1\S001-038.jpg | 0.644069 | yes |
| D:\desktop\image1\S001-040.jpg | 0.287234 | |
| D:\desktop\image1\S001-041.jpg | 0.284123 | |
| D:\desktop\image1\S004-001.jpg | 0.347806 | |
| D:\desktop\image1\S004-002.jpg | 0.300212 | |
| D:\desktop\image1\S004-003.jpg | 0.362331 | |
| D:\desktop\image1\S004-004.jpg | 0.378441 | |
| D:\desktop\image1\S004-005.jpg | 0.351368 | |
| D:\desktop\image1\S004-116.jpg | 0.347412 | |
| D:\desktop\image1\S014-001.jpg | 0.438328 | |
| D:\desktop\image1\S014-002.jpg | 0.602056 | yes |
| D:\desktop\image1\S014-003.jpg | 0.45454 | |
| D:\desktop\image1\S014-004.jpg | 0.555613 | yes |
| D:\desktop\image1\S018-002.jpg | 0.300771 | |
| D:\desktop\image1\S018-003.jpg | 0.288213 | |
| D:\desktop\image1\S018-004.jpg | 0.310391 | |
| D:\desktop\image1\S018-005.jpg | 0.293818 | |
| D:\desktop\image1\S018-006.jpg | 0.270315 | |

基于欧氏距离度量的 Gabor 图像检索算法的实验结果如表 7-6 所示。

**表 7-6　基于欧氏距离度量的 Gabor 图像检索算法的实验结果**

| | | |
|---|---|---|
| D:\desktop\image1\S001-001.jpg | 63230.3 | |
| D:\desktop\image1\S001-002.jpg | 0 | yes |
| D:\desktop\image1\S001-003.jpg | 55957.7 | yes |
| D:\desktop\image1\S001-004.jpg | 45162.8 | yes |
| D:\desktop\image1\S001-005.jpg | 47509.7 | yes |

续表

| | | |
|---|---|---|
| D:\desktop\image1\S001-006.jpg | 45944 | yes |
| D:\desktop\image1\S001-007.jpg | 61168.1 | yes |
| D:\desktop\image1\S001-008.jpg | 51290.1 | yes |
| D:\desktop\image1\S001-009.jpg | 53291.9 | yes |
| D:\desktop\image1\S001-010.jpg | 58427.5 | yes |
| D:\desktop\image1\S001-011.jpg | 53169.7 | yes |
| D:\desktop\image1\S001-012.jpg | 51439.6 | yes |
| D:\desktop\image1\S001-013.jpg | 58753.5 | yes |
| D:\desktop\image1\S001-014.jpg | 57585.8 | yes |
| D:\desktop\image1\S001-015.jpg | 48597.2 | yes |
| D:\desktop\image1\S001-016.jpg | 46975.2 | yes |
| D:\desktop\image1\S001-017.jpg | 51819.9 | yes |
| D:\desktop\image1\S001-018.jpg | 49463.6 | yes |
| D:\desktop\image1\S001-019.jpg | 57329.2 | yes |
| D:\desktop\image1\S001-020.jpg | 53077.4 | yes |
| D:\desktop\image1\S001-021.jpg | 51522 | yes |
| D:\desktop\image1\S001-022.jpg | 58207.1 | yes |
| D:\desktop\image1\S001-023.jpg | 37890.1 | yes |
| D:\desktop\image1\S001-024.jpg | 42270.3 | yes |
| D:\desktop\image1\S001-025.jpg | 50177.8 | yes |
| D:\desktop\image1\S001-026.jpg | 50630.7 | yes |
| D:\desktop\image1\S001-027.jpg | 37875 | yes |
| D:\desktop\image1\S001-028.jpg | 46547.1 | yes |
| D:\desktop\image1\S001-029.jpg | 53994.5 | yes |
| D:\desktop\image1\S001-030.jpg | 58141.8 | yes |
| D:\desktop\image1\S001-031.jpg | 53753.5 | yes |
| D:\desktop\image1\S001-032.jpg | 47161 | yes |
| D:\desktop\image1\S001-033.jpg | 49515.9 | yes |
| D:\desktop\image1\S001-034.jpg | 47459.8 | yes |
| D:\desktop\image1\S001-035.jpg | 57012.8 | yes |
| D:\desktop\image1\S001-036.jpg | 56715.8 | yes |
| D:\desktop\image1\S001-037.jpg | 55855.6 | yes |
| D:\desktop\image1\S001-038.jpg | 47723.6 | yes |
| D:\desktop\image1\S001-040.jpg | 65382.5 | |
| D:\desktop\image1\S001-041.jpg | 65437.5 | |
| D:\desktop\image1\S004-001.jpg | 75407.8 | |

续表

| | | |
|---|---|---|
| D:\desktop\image1\S004-002.jpg | 75474 | |
| D:\desktop\image1\S004-003.jpg | 74261.8 | |
| D:\desktop\image1\S004-004.jpg | 72958.3 | |
| D:\desktop\image1\S004-005.jpg | 73284.2 | |
| D:\desktop\image1\S004-116.jpg | 73490.3 | |
| D:\desktop\image1\S014-001.jpg | 59005.5 | yes |
| D:\desktop\image1\S014-002.jpg | 56865.2 | yes |
| D:\desktop\image1\S014-003.jpg | 57641.2 | yes |
| D:\desktop\image1\S014-004.jpg | 54348.1 | yes |
| D:\desktop\image1\S018-002.jpg | 64542.9 | |
| D:\desktop\image1\S018-003.jpg | 65608.2 | |
| D:\desktop\image1\S018-004.jpg | 65511.3 | |
| D:\desktop\image1\S018-005.jpg | 62970.3 | |
| D:\desktop\image1\S018-006.jpg | 64321.3 | |

### 3. 实验分析

从实验结果中看出，该算法对图像的涂抹、覆盖很敏感。但是整体上，匹配到的相似图像的成效尚可。基于余弦相似度度量的 Gabor 算法的阈值为 0.50，查准率为 29/31×100%=93.5%，查全率为 29/40×100%=72.5%。基于欧氏距离度量的 Gabor 算法的阈值为 62000，查准率为 37/41×100%=90.2%，查全率为 37/40×100%=92.5%。

## 7.11 HOG 算法

HOG，英文全称为 Locally Normalised Histogram of Gradient Orientation in Dense Overlapping Grids，即局部归一化的梯度方向直方图，是一种对图像局部重叠区域的密集型描述符，它统计图像的局部区域的梯度方向信息作为局部凸显该区域的表征。具体来说，将梯度方向（0°～360°）划分成 9 个区间，将图像划分成 16×16 像素的若干块（Block），每个块再划分成 4 个 8×8 像素的单元格（Cell）。对于每一个 Cell，计算每一像素点的梯度方向和模，按梯度方向增加对应 bin 的值，最终综合 $n$ 个 Cell 的梯度直方图形成一个高维描述子向量[13-16]。

### 7.11.1　HOG 算法的使用

在 Windows 7 环境下，安装 CodeBlock 编译环境，并配置 OpenCV。

该算法对应的项目名称是 HoG_Project。具体执行步骤如下。

（1）打开 HoG_Project 文件夹，然后用 CodeBlocks 打开 HoG_Project.cbp 文件。

（2）如果不修改输入数据及相关参数，则直接执行（4）。

（3）如果修改数据，则可在主函数中的第一行代码中进行修改：

```
Mat trainImgSetDescri=readImg("D:\\desktop\\HoG_Project\\ image150x
200","jpg");
```

其中，readImg 的第一个参数是图像集的文件名，第二个参数是图像文件的后缀名。

（4）运行 HoG.cpp 文件即可。

注意：本实验中使用的图像为灰度图，尺寸大小都是相同的。

### 7.11.2　HOG 算法原理

在 OpenCV 中，HOG 描述子是针对一个检测窗口而言的，一个窗口有[(128−16)/8+1]×[(64−16)/8+1]=105 个 Block，一个 Block 有 4 个 Cell，一个 Cell 的 HOG 描述子向量的长度是 9，所以一个检测窗口的 HOG 描述子的向量长度是 105×4×9=3780 维。文献[15]中给出了 HOG 算法原理的步骤：

（1）灰度化图像处理。在本实验中，图像集是灰度图，所以该算法代码就没有再对图像一一进行灰度化处理。

（2）计算图像每一个像素的梯度，该梯度包括大小和方向。计算图像横坐标和纵坐标的梯度，在此基础上求每个像素位置的梯度方向值。这一步不仅可以捕获轮廓和纹理信息，还可以进一步弱化光照的影响。

（3）将图像分割成小的 Cell。一个 Cell 是 HOG 特征最小的结构单位，Block 和窗口 win 的滑动步长就是一个 Cell 的宽度或高度，因此，需要先把整幅图像分割成一个个的 Cell。

（4）为每个 Cell 构建梯度方向直方图。这是最关键的一步，它主要是统计局部图像梯度信息并进行量化，得到局部图像区域的特征描述向量。

（5）将 Cell 组成大的 Block，然后 Block 内归一化梯度直方图。这也是很关键的一步。

（6）生成 HOG 特征描述向量。将所有的 Block 的 HOG 描述符组合在一起，形成最终的特征向量，该特征向量描述了检测窗口的图像内容。

在本实验中，我们使用的图像集全部都是灰度图，尺寸大小都是 150×200 像素。在本实验中检测窗口的大小为 64×128 像素（其中 Block 大小为 16×16 像素，Block 的滑动步长为 8×8 像素，Cell 大小为 8×8 像素，一个 Cell 内统计 9 个方向的梯度直方图），窗口滑动步长为 64×64 像素，扩充像素数为 42×56 像素。也就是说，宽扩充 42 个像素点，高扩充 56 个像素点，因此每一幅图像的检测窗口共有 9 个，即我们使用 3780×9=34 020 维度的特征向量表示每一幅图像。于是，我们便得到了每一幅图像的表示，接着进行图像之间的匹配，本算法中我们使用的是欧氏距离度量方法。阈值为 16，也就是说，我们认为欧氏距离小于 16 的图像都是相似图像。

### 7.11.3 HOG 算法实现

我们使用的是 OpenCV 中的 HOG 特征提取功能，它使用了 HOGDescriptor 类的封装。下面程序中的部分代码来自文献[13-16]。

```
Mat obtainHoG(char* imgFileame )
{
    HOGDescriptor *hog1= new HOGDescriptor(cvSize(64, 128),
cvSize(16, 16), cvSize(8, 8), cvSize(8, 8), 9);
     //cvSize(64, 128):检测窗口的大小
     // cvSize(16, 16):块大小
     //cvSize(8, 8):块的滑动步长
     //cvSize(8, 8):单元格的大小
     //9:直方图 bin 的数量

    vector<float> ders1;
    vector<Point>locs1;
//This function computes the hog features for you
    Mat trainImg1; //需要分析的图片
    trainImg1=imread(imgFileame,1);   //读取图像
hog1->compute(trainImg1,ders1,Size(64,64),Size(42,56),locs1);
```

```
    //The HOG features computed for grayImg are stored in ders vector
to make it into a matrix which can be used for training later use the
following for loop
    //trainImg1:源图像，只支持 CV_8UC1 和 CV_8UC4 数据类型
    //ders1:descriptors:  返回的 HOG 特征向量，descriptors.size 是 HOG 特征
的维数
    //Size(64,64):窗口移动步长
    //Size(42,56):扩充像素数
    //locs1: 对于正样本可以直接取(0,0),负样本为随机产生合理坐标范围内的点坐标

        Mat Hogfeat1;
        Hogfeat1.create(ders1.size(),1,CV_32FC1);
        for(unsigned int i=0; i<ders1.size(); i++)
        {
            Hogfeat1.at<float>(i,0)=ders1.at(i);
        }
        return Hogfeat1;
    }

    //主函数
    int main()
    {
        Mat trainImgSetDescri=readImg("D:\\desktop\\HoG_Project\\image
150x200","jpg");
        cout<<"--------------------"<<endl;
        cout<<trainImgSetDescri.rows<<"   "<<trainImgSetDescri.cols<<
endl;

        ofstream outfile("HoG_distance.txt");
        Mat baseDescri=trainImgSetDescri.row(0);
        for(int i=0;i<trainImgSetDescri.rows;i++)
        {
            Mat targetDescri=trainImgSetDescri.row(i);
            double distance=calculateEuDis(targetDescri,baseDescri);
            if(distance<16)
                outfile<<"与第"<<i+1<<"个图像的距离是"<<distance<<"yes"
<<endl;
```

```
        else
            outfile<<"与第"<<i+1<<"个图像的距离是"<<distance<<endl;
    }
    return 0;
}
```

### 7.11.4 HOG 算法的实验数据、实验结果及分析

#### 1. 实验数据

实验输入数据为 150×200 像素的原始图像，数据集路径为“..\HoG_Project\image150×200”。输入的测试图像为文件夹中的第一个图像。实验输出数据为图像的序号和图像之间的距离，输出数据存于“.\HoG_Project\HoG_distance.txt”。

#### 2. 实验结果

查询图像如图 7-17 所示。



图 7-17 查询图像（7）

检索出的部分相似图像如图 7-18 所示。



图 7-18 检索出的部分相似图像（7）

以下是 HoG_distance.txt 的部分内容展示，如表 7-7 所示。

表 7-7　HoG_distance.txt 部分内容展示

| | | |
|---|---|---|
| 与第 1 个图像的距离是 | 0 | yes |
| 与第 2 个图像的距离是 | 7.49834 | yes |
| 与第 3 个图像的距离是 | 11.3776 | yes |
| 与第 4 个图像的距离是 | 15.6236 | yes |
| 与第 5 个图像的距离是 | 13.2175 | yes |
| 与第 6 个图像的距离是 | 14.0539 | yes |
| 与第 7 个图像的距离是 | 15.4788 | yes |
| 与第 8 个图像的距离是 | 14.6722 | yes |
| 与第 9 个图像的距离是 | 12.071 | yes |
| 与第 10 个图像的距离是 | 13.8017 | yes |
| 与第 11 个图像的距离是 | 13.875 | yes |
| 与第 12 个图像的距离是 | 14.1833 | yes |
| 与第 13 个图像的距离是 | 15.7912 | yes |
| 与第 14 个图像的距离是 | 13.657 | yes |
| 与第 15 个图像的距离是 | 16.0576 | |
| 与第 16 个图像的距离是 | 17.8302 | |
| 与第 17 个图像的距离是 | 17.6491 | |
| 与第 18 个图像的距离是 | 14.3837 | yes |
| 与第 19 个图像的距离是 | 17.4082 | |
| 与第 20 个图像的距离是 | 13.2531 | yes |
| 与第 21 个图像的距离是 | 12.9948 | yes |
| 与第 22 个图像的距离是 | 14.2158 | yes |
| 与第 23 个图像的距离是 | 12.8095 | yes |
| 与第 24 个图像的距离是 | 12.9387 | yes |
| 与第 25 个图像的距离是 | 12.9261 | yes |
| 与第 26 个图像的距离是 | 14.0139 | yes |
| 与第 27 个图像的距离是 | 14.366 | yes |
| 与第 28 个图像的距离是 | 14.4572 | yes |
| 与第 29 个图像的距离是 | 14.0505 | yes |
| 与第 30 个图像的距离是 | 14.6448 | yes |
| 与第 31 个图像的距离是 | 14.3295 | yes |
| 与第 32 个图像的距离是 | 15.432 | yes |

续表

| | | |
|---|---|---|
| 与第 33 个图像的距离是 | 16.0668 | |
| *与第 34 个图像的距离是* | *13.5364* | *yes* |
| 与第 35 个图像的距离是 | 13.5504 | yes |
| 与第 36 个图像的距离是 | 15.1888 | yes |
| 与第 37 个图像的距离是 | 14.4207 | yes |
| 与第 38 个图像的距离是 | 13.3945 | yes |
| 与第 39 个图像的距离是 | 12.9389 | yes |
| 与第 40 个图像的距离是 | 13.4662 | yes |
| 与第 41 个图像的距离是 | 2.83757 | yes |
| 与第 42 个图像的距离是 | 3.94401 | yes |
| 与第 43 个图像的距离是 | 18.5763 | |
| 与第 44 个图像的距离是 | 18.0502 | |
| 与第 45 个图像的距离是 | 19.2137 | |
| 与第 46 个图像的距离是 | 19.052 | |
| 与第 47 个图像的距离是 | 18.02 | |
| 与第 48 个图像的距离是 | 18.2568 | |
| 与第 49 个图像的距离是 | 18.1667 | |
| 与第 50 个图像的距离是 | 18.5785 | |
| 与第 51 个图像的距离是 | 18.6432 | |
| 与第 52 个图像的距离是 | 19.4722 | |
| 与第 53 个图像的距离是 | 18.3845 | |
| 与第 54 个图像的距离是 | 17.7761 | |
| 与第 55 个图像的距离是 | 17.9385 | |
| 与第 56 个图像的距离是 | 18.8051 | |
| 与第 57 个图像的距离是 | 18.3599 | |
| 与第 58 个图像的距离是 | 18.5855 | |

### 3. 实验分析

从检索结果中我们发现，返回结果并不多，但是结果中的误报数很小（为零），所有返回的图像都与查询相关。本次实验的查准率是 37/37×100%=100%，查全率是 37/42×100%=88.095%。

## 7.12　深度学习特征

深度学习和传统模式识别方法的最大不同在于它是从大数据中自动学习特征，而非采用手工设计的特征。好的特征可以在很大程度上提高模式识别系统的性能。深度学习运用于物体识别领域具有重大突破意义的就是人脸识别。无论是传统的人脸检测和识别，还是基于深度学习的检测和识别，面临的最大挑战都是如何区分由于光线、姿态和表情等因素引起的类内变化和由于身份不同产生的类间变化。这两种变化分布是非线性的且极为复杂，传统的现行模型无法将它们有效地分开。深度学习的目的是将各种因素成功地分开，例如在深度模型的最后一个隐含层，不同的神经元代表了不同的因素。如果将这个隐含层当作特征表示，人脸识别、姿态估计、表情识别、年龄估计就会变得十分简单，因此通过多层的非线性变换得到新的简单线性特征表示。这些特征就尽可能多地去掉类内变化，而保留类间变化。人脸识别有两类任务，即人脸确认和人脸辨识。人脸确认的任务是判断两张照片是否为同一个人，属二分类问题，随机猜中的概率为 50%。人脸辨识的任务是将一张人脸图像准确地分到 $N$ 个类别之一内，类别是由人脸的身份定义的。这是一个多类问题，更具有挑战性，其难度随着类别数的增多而增大，随机猜中的概率为 $1/N$。这两类任务都可以通过深度模型学习人脸的特征表达。

### 7.12.1　深度学习算法的使用

本部分的特征是使用 Ubuntu 上的 Caffe 工具提取的。

算法对应的项目文件名称为“deepLearningSearch.sln”，位于“../deepLearningSearch”路径下。

打开 deepLearningSearch 文件夹，使用 Visual Studio 2010 打开“deepLearningSearch.sln”，运行“deepLearningSearch.cpp”即可。

### 7.12.2　深度学习算法原理

我们使用 ImageNet 网络结构，主要包括 5 个卷积层、3 个全连接层、下采样

层（Pooling 层）、线性修正单元（ReLU 层）以及标志所属类别的输出层[17]。网络最后一个全连接层的输出需要设置为类别数，输入图片尺寸为 227×227 像素，输出为属于每个类别的概率值。我们选择网络结构中的倒数第二个全连接层作为要提取的深度特征层，由于该层后跟有线性修正单元（ReLU 层），所以提取出的深度特征向量都是非负值。为了防止过拟合现象，网络中使用了 Drop-Out 层，过滤学习到的一些冗余信息。

使用 ImageNet 训练模型，我们选择网络结构中的倒数第二个全连接层作为要提取的深度特征层，因此对于每张图片，都能得到一个 4096 维的特征向量，所以每张图片我们可以获得的特征向量维度为 4096。

### 7.12.3 深度学习算法实现

算法步骤：该部分对应的代码包含在源码中的 extract_feature 文件夹下，geetDeepID.m 文件可以实现对指定文件列表中的所有图片进行提取深度特征。

### 7.12.4 深度学习算法的实验数据、实验结果及分析

#### 1. 实验数据

图像库中为利用深度学习得到的 497（行）×128（列）或者 497（行）×4096（列）的数据。

查询图像的深度特征为 1（行）×128（列）。

#### 2. 实验结果

实验结果在与“deepLearningSearch.cpp”同一目录下的“similityDLFeature.txt”文件中。每行中第一列代表图像的编号，第二列代表查询图像与图像库中的当前图像的余弦相似度。

查询图像如图 7-19 所示。



图 7-19　查询图像（8）

根据余弦相似度的阈值，检索出的部分相似图像如图 7-20 所示。



图 7-20　检索出的部分相似图像（8）

表 7-8 所示为基于 4096 维度的余弦相似度度量的深度学习算法之图像检索的结果。

表 7-8　图像检索结果

| | | | |
|---|---|---|---|
| image/S001-001 (2).jpg | 0 | image/S004-058.jpg | 0.646893 |
| image/S001-001(1).jpg | | image/S004-059.jpg | 0.634859 |
| image/S001-001.jpg | 1 | image/S004-060.jpg | 0.665317 |
| image/S001-002.jpg | | image/S004-061.jpg | 0.545099 |
| image/S001-003.jpg | 0.922682 | image/S004-062.jpg | 0.582325 |
| image/S001-004.jpg | 0.900318 | image/S004-063.jpg | 0.600629 |
| image/S001-005.jpg | 0.85459 | image/S004-064.jpg | 0.622065 |
| image/S001-006.jpg | 0.908704 | image/S004-065.jpg | 0.606282 |
| image/S001-007.jpg | 0.899554 | image/S004-066.jpg | 0.591947 |
| image/S001-008.jpg | 0.898334 | image/S004-067.jpg | 0.62951 |
| image/S001-009.jpg | 0.885297 | image/S004-068.jpg | 0.63104 |

续表

| | | | |
|---|---|---|---|
| image/S001-010.jpg | 0.882518 | image/S004-069.jpg | 0.647492 |
| image/S001-011.jpg | 0.865176 | image/S004-070.jpg | 0.644658 |
| image/S001-012.jpg | 0.855715 | image/S004-071.jpg | 0.638584 |
| image/S001-013.jpg | 0.868195 | image/S004-072.jpg | 0.608694 |
| image/S001-014.jpg | 0.874543 | image/S004-073.jpg | 0.610534 |
| image/S001-015.jpg | 0.807534 | image/S004-074.jpg | 0.62731 |
| image/S001-016.jpg | 0.773938 | image/S004-075.jpg | 0.609546 |
| image/S001-017.jpg | 0.859363 | image/S004-076.jpg | 0.646584 |
| image/S001-018.jpg | 0.848863 | image/S004-077.jpg | 0.642922 |
| image/S001-019.jpg | 0.849349 | image/S004-078.jpg | 0.639534 |
| image/S001-020.jpg | 0.837319 | image/S004-079.jpg | 0.627845 |
| image/S001-021.jpg | 0.837682 | image/S004-080.jpg | 0.62548 |
| image/S001-022.jpg | 0.827605 | image/S004-081.jpg | 0.620852 |
| image/S001-023.jpg | 0.803921 | image/S004-082.jpg | 0.628034 |
| image/S001-024.jpg | 0.815563 | image/S004-083.jpg | 0.617528 |
| image/S001-025.jpg | 0.826165 | image/S004-084.jpg | 0.615985 |
| image/S001-026.jpg | 0.837509 | image/S004-085.jpg | 0.637718 |
| image/S001-027.jpg | 0.850452 | image/S004-086.jpg | 0.638943 |
| image/S001-028.jpg | 0.845345 | image/S004-087.jpg | 0.645529 |
| image/S001-029.jpg | 0.833506 | image/S004-088.jpg | 0.653869 |
| image/S001-030.jpg | 0.843088 | image/S004-089.jpg | 0.646633 |
| image/S001-031.jpg | 0.868124 | image/S004-090.jpg | 0.652496 |
| image/S001-032.jpg | 0.835493 | image/S004-091.jpg | 0.612805 |
| image/S001-033.jpg | 0.8457 | image/S004-092.jpg | 0.623109 |
| image/S001-034.jpg | 0.849687 | image/S004-093.jpg | 0.631979 |
| image/S001-035.jpg | 0.776568 | image/S004-094.jpg | 0.631728 |
| image/S001-036.jpg | 0.804949 | image/S004-095.jpg | 0.612774 |
| image/S001-037.jpg | 0.779202 | image/S004-096.jpg | 0.61645 |
| image/S001-038.jpg | 0.777569 | image/S004-097.jpg | 0.63586 |
| image/S001-039.jpg | 0.831507 | image/S004-098.jpg | 0.636123 |
| image/S001-040.jpg | 0.846515 | image/S004-099.jpg | 0.635838 |
| image/S001-041.jpg | 0.846174 | image/S004-100.jpg | 0.628445 |
| image/S004-001.jpg | 0.910841 | image/S004-101.jpg | 0.608144 |
| image/S004-002.jpg | 0.699536 | image/S004-102.jpg | 0.605691 |
| image/S004-003.jpg | 0.668362 | image/S004-103.jpg | 0.588585 |
| image/S004-004.jpg | 0.673445 | image/S004-104.jpg | 0.58804 |

续表

| | | | |
|---|---|---|---|
| image/S004-005.jpg | 0.654261 | image/S004-105.jpg | 0.617735 |
| image/S004-006.jpg | 0.64821 | image/S004-106.jpg | 0.631 |
| image/S004-007.jpg | 0.642054 | image/S004-107.jpg | 0.59041 |
| image/S004-008.jpg | 0.656356 | image/S004-108.jpg | 0.568613 |
| image/S004-009.jpg | 0.667077 | image/S004-109.jpg | 0.566592 |
| image/S004-010.jpg | 0.644788 | image/S004-110.jpg | 0.597622 |
| image/S004-011.jpg | 0.646769 | image/S004-111.jpg | 0.568693 |
| image/S004-012.jpg | 0.637414 | image/S004-112.jpg | 0.583867 |
| image/S004-013.jpg | 0.647383 | image/S004-113.jpg | 0.588843 |
| image/S004-014.jpg | 0.65251 | image/S004-114.jpg | 0.592273 |
| image/S004-015.jpg | 0.649314 | image/S004-115.jpg | 0.62327 |
| image/S004-016.jpg | 0.662375 | image/S004-116.jpg | 0.627604 |
| image/S004-017.jpg | 0.671773 | image/S014-001.jpg | 0.747208 |
| image/S004-018.jpg | 0.673133 | image/S014-002.jpg | 0.71872 |
| image/S004-019.jpg | 0.665108 | image/S014-003.jpg | 0.749315 |
| image/S004-020.jpg | 0.662304 | image/S014-004.jpg | 0.757009 |
| image/S004-021.jpg | 0.657209 | image/S014-005.jpg | 0.742369 |
| image/S004-022.jpg | 0.654871 | image/S014-006.jpg | 0.711628 |
| image/S004-023.jpg | 0.672706 | image/S014-007.jpg | 0.695924 |
| image/S004-024.jpg | 0.668303 | image/S014-008.jpg | 0.712512 |
| image/S004-025.jpg | 0.656387 | image/S014-009.jpg | 0.719256 |
| image/S004-026.jpg | 0.68523 | image/S014-010.jpg | 0.676167 |
| image/S004-027.jpg | 0.620429 | image/S014-011.jpg | 0.704337 |
| image/S004-028.jpg | 0.62296 | image/S014-012.jpg | 0.65972 |
| image/S004-029.jpg | 0.676022 | image/S014-013.jpg | 0.687088 |
| image/S004-030.jpg | 0.678142 | image/S014-014.jpg | 0.688586 |
| image/S004-031.jpg | 0.64587 | image/S014-015.jpg | 0.703945 |
| image/S004-032.jpg | 0.664092 | image/S014-016.jpg | 0.673978 |
| image/S004-033.jpg | 0.659842 | image/S014-017.jpg | 0.663239 |
| image/S004-034.jpg | 0.648629 | image/S014-018.jpg | 0.666403 |
| image/S004-035.jpg | 0.647925 | image/S014-019.jpg | 0.68505 |
| image/S004-036.jpg | 0.659833 | image/S014-020.jpg | 0.668687 |
| image/S004-037.jpg | 0.639311 | image/S014-021.jpg | 0.709343 |
| image/S004-038.jpg | 0.63661 | image/S014-022.jpg | 0.69544 |
| image/S004-039.jpg | 0.612968 | image/S014-023.jpg | 0.711566 |

续表

| | | | |
|---|---|---|---|
| image/S004-040.jpg | 0.615646 | image/S014-024.jpg | 0.713596 |
| image/S004-041.jpg | 0.687405 | image/S014-025.jpg | 0.692125 |
| image/S004-042.jpg | 0.657779 | image/S014-026.jpg | 0.678836 |
| image/S004-043.jpg | 0.620975 | image/S014-027.jpg | 0.66129 |
| image/S004-044.jpg | 0.616763 | image/S014-028.jpg | 0.663964 |
| image/S004-045.jpg | 0.674532 | image/S014-029.jpg | 0.731745 |
| image/S004-046.jpg | 0.680095 | image/S014-030.jpg | 0.730798 |
| image/S004-047.jpg | 0.68329 | image/S014-031.jpg | 0.727353 |
| image/S004-048.jpg | 0.676631 | image/S014-032.jpg | 0.741873 |
| image/S004-049.jpg | 0.640609 | image/S014-033.jpg | 0.699338 |
| image/S004-050.jpg | 0.645189 | image/S014-034.jpg | 0.706374 |
| image/S004-051.jpg | 0.639477 | image/S014-035.jpg | 0.671856 |
| image/S004-052.jpg | 0.6273 | image/S014-036.jpg | 0.704711 |
| image/S004-053.jpg | 0.638634 | image/S014-037.jpg | 0.698981 |
| image/S004-054.jpg | 0.648569 | image/S014-038.jpg | 0.749163 |
| image/S004-055.jpg | 0.672069 | image/S014-039.jpg | 0.742598 |
| image/S004-056.jpg | 0.679226 | image/S014-040.jpg | 0.700474 |
| image/S004-057.jpg | 0.632876 | image/S014-041.jpg | 0.706653 |
| | 0.62777 | | |
| | 0.644625 | | |

3．实验分析

128 维度时：相似图像的余弦相似度阈值为 0.74。

4096 维度时：相似图像的余弦相似度阈值为 0.773 以上。

最后，使用深度学习算法进行人脸检索的查准率为 41/43×100%=95.35%，查全率为 41/42×100%=97.62%。

## 参考文献

[1] 欧氏距离. 百度百科. http://baike.baidu.com/.

[2] 余弦相似度. 百度百科. http://baike.baidu.com/.

[3] 感知哈希算法 PHash 的 OpenCV 实现. http://blog.csdn.net/eternity1118_/article/details/51088019.

[4] 主成分分析. http://blog.csdn.net/dahuacai/article/details/50503246.

[5] PCA 的数学原理. http://blog.codinglabs.org/articles/pca-tutorial.html.

[6] 初识 PCA 数据降维. http://www.cnblogs.com/hxsyl/p/4548264.html.

[7] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. IJCV2004. http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf.

[8] OpenCV, BoF-SIFT. http://www.codeproject.com/Articles/619039/Bag-of-Features-Descriptor-on-SIFT- Features-with-O.

[9] FLANN. https://github.com/jlblancoc/nanoflann.

[10] Gabor 滤波器学习. http://blog.csdn.net/jinshengtao/article/details/17797641.

[11] Gabor filter. Wikipedia. https://en.wikipedia.org/wiki/Gabor_filter.

[12] 图像的 Gabor 小波变换. http://download.csdn.net/download/CPFZZ/1976739.

[13] OpenCV—HoG 特征. http://blog.csdn.net/loadstar_kun/article/details/7728463.

[14] 利用 OpenCV 提取 HOG 特征需要了解的参数. http://blog.sciencenet.cn/blog-702148-762019.html.

[15] HOG：从理论到 OpenCV 实践. http://www.tuicool.com/articles/fM3Mvu.

[16] Extracting HoG Features using OpenCV. http://stackoverflow.com/questions/11626140/extracting-hog-features-using-opencv.

[17] ImageNet 深度卷积神经网络笔记. http://www.gageet.com/2014/09140.php.

# 第 8 章

# 人脸检测商业软件及其应用示例

本章介绍两款知名的人脸检测商业软件，分别是 VeriLook 和 Face++。除这两款商业软件外，OpenCV 是一款非常常用的免费人脸识别软件，得到了计算机视觉领域研究人员和工程师的青睐。由于本书在前面章节中已经详细讲解了如何利用 OpenCV 进行人脸检测、人脸识别和人脸检索，在本章中，我们不再单独讲解。

## 8.1 人脸检测商业软件之 VeriLook

Neurotechnology[1] 是一家总部位于欧洲（立陶宛）的公司，它提供了指纹、人脸、虹膜、声音和掌印的识别算法和软件开发 SDK，它的产品被用在许多应用程序中，如边境口岸、刑事调查、选民登记系统等。它的人脸检测和识别应用可以实时地处理输入的图片和视频，并且检测效果良好、稳定。

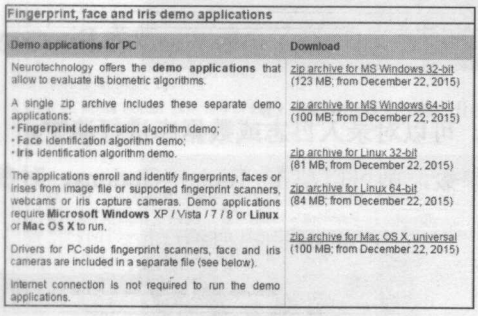
Neurotechnology 中人脸检测和识别的软件产品为 VeriLook。这里我们介绍一

下如何使用 VeriLook 进行人脸检测，步骤如下。

（1）VeriLook 软件下载地址为：http://www.neurotechnology.com/download.html# demo，打开链接后，页面如图 8-1 所示。

选择和自己计算机相匹配的版本，本实验使用的是：zip archive for MS Windows 64-bit，下载后对应的文件名为：Neurotec_Biometric_6_0_Algorithm_Demo_Win64_x64_2015-12-22。

（2）解压上述文件，运行“FacesAlgorithmDemo.exe”后，显示主界面，如图 8-2 所示。



Fingerprint, face and iris demo applications

| Demo applications for PC | Download |
| --- | --- |
| Neurotechnology offers the **demo applications** that allow to evaluate its biometric algorithms. | zip archive for MS Windows 32-bit (123 MB; from December 22, 2015) |
| A single zip archive includes these separate demo applications:<br>• **Fingerprint** identification algorithm demo;<br>• **Face** identification algorithm demo;<br>• **Iris** identification algorithm demo. | zip archive for MS Windows 64-bit (100 MB; from December 22, 2015) |
| | zip archive for Linux 32-bit (81 MB; from December 22, 2015) |
| The applications enroll and identify fingerprints, faces or irises from image file or supported fingerprint scanners, webcams or iris capture cameras. Demo applications require **Microsoft Windows** XP / Vista / 7 / 8 or **Linux** or **Mac OS X** to run. | zip archive for Linux 64-bit (84 MB; from December 22, 2015) |
| | zip archive for Mac OS X, universal (100 MB; from December 22, 2015) |
| Drivers for PC-side fingerprint scanners, face and iris cameras are included in a separate file (see below). | |
| Internet connection is not required to run the demo applications. | |

图 8-1　VeriLook 下载页面

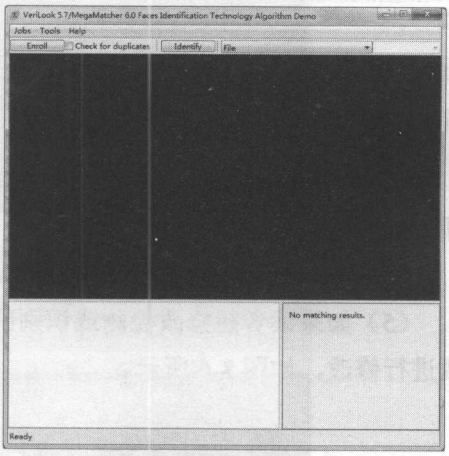



图 8-2　VeriLook 的主界面

单击 Enroll 按钮，选择一张图片后，该软件可以提取人脸特征，标记出人脸对应位置，并将检测结果存入数据库中，Verilook 主界面左下方的面板中会显示录入的日志。

单击 Identify 按钮，选择一张图片后，该软件在数据库查找和这张新图片相匹配的图片，VeriLook 主界面右下方的面板中会显示匹配的结果。

（3）单击 Enroll 按钮，选择一张图片后，界面如图 8-3 所示。检测到的人脸同时存入数据库。单击 Identify 按钮，选择一张图片后，界面如图 8-4 所示。

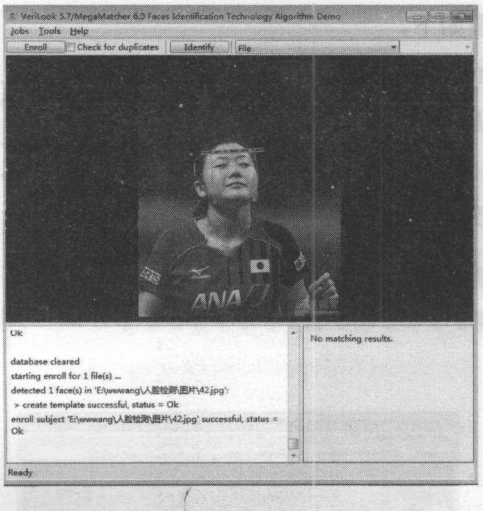



图 8-3　单击 Enroll 按钮选择一张图片后的界面




图 8-4　单击 Identify 按钮选择一张图片后的界面

（4）如果要录入或识别一个文件夹内的所有文件，则只需将 File 改为 Directory 即可，如图 8-5 所示。

单击 Tools→clear Log/clear database，可以对录入日志或数据库进行清空。

（5）如果读者想修改检测或识别的参数，则可以单击 Tools→Options，对参数进行修改，如图 8-6 所示。

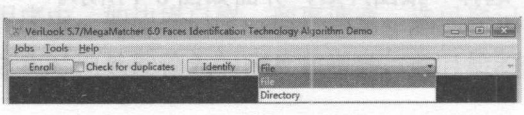



图 8-5　切换 File 和 Directory

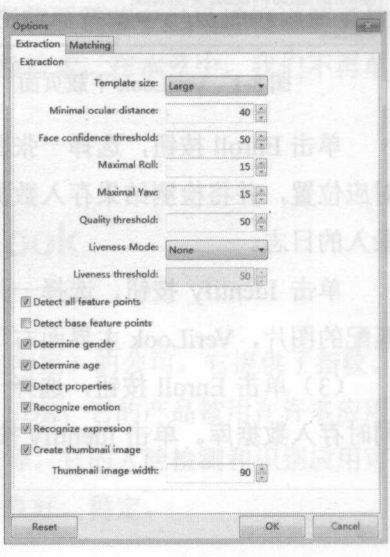
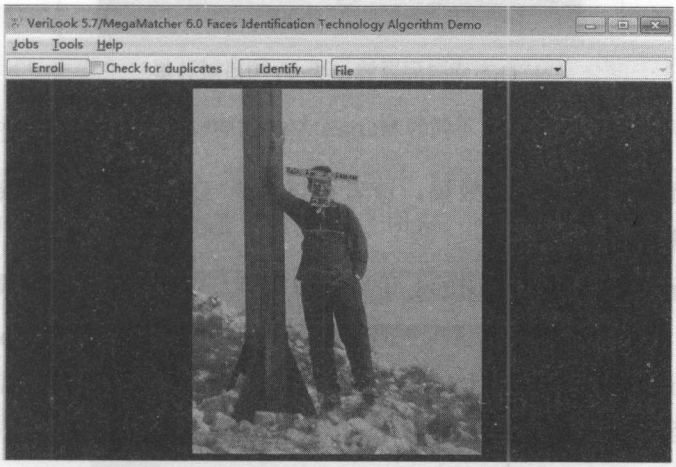



图 8-6　参数调整对话框

Minimal ocular distance，即人脸模板的两眼间的最小距离，默认值为 40，最小值为 8。当需要检测远景镜头中的人脸时，可以调整这个值。图 8-7（a）、（b）所示是不同的 Minimal ocular distance 值（其他参数值相同）的检测效果。




（a）Minimal ocular distance 为 40 的检测效果




（b）Minimal ocular distance 为 8 的检测效果

图 8-7　不同的 Minimal ocular distance 值的检测效果

Maximal Yaw 是最大偏移角度（绕 $y$ 轴），默认值是 15。当需要检测侧脸时，我们可以调整 Maximal Yaw 的值。图 8-8（a）、（b）所示是不同的 Maximal Yaw 值（其他参数值一致）的检测效果。




（a）Maximal Yaw 为 15 的检测效果




（b）Maximal Yaw 为 90 的检测效果

图 8-8　不同的 Maximal Yaw 值的检测效果

关于其他参数的含义和效果，读者可以参考 Documentation\FacesAlgorithm Demo.pdf 文档。

上面展示了如何利用 VeriLook 进行人脸检测、识别，以及如何调整参数。关于 VeriLook 的更多使用，读者可查看 Neurotechnology 的网站和 Documentation\FacesAlgorithmDemo.pdf 文档。总之，VeriLook 是一款非常高效、实用的人脸检测软件，得到了来自全世界的多种不同用户的推荐。

## 8.2　人脸检测商业软件之 Face++

Face++[2]是北京旷视科技有限公司旗下的新视觉服务平台，Face++提供的产品服务有：人脸检测——人脸检测和关键点检测；人脸分析——微笑分析，性别、

年龄、种族分析；人脸识别——1:1 人脸验证、1:*N* 人脸识别和大规模人脸检索。它非常容易使用，检测效果好、时间短，是一款强大、易用的商用软件。

Face++中提供了多种 SDK，本书以 Face++的 MATLAB SDK 为例，介绍一下如何使用 Face++进行人脸检测，具体步骤如下。

（1）打开 Face++网站（http://www.faceplusplus.com.cn/）。

（2）单击右上角的“注册”按钮，如图 8-9 所示。

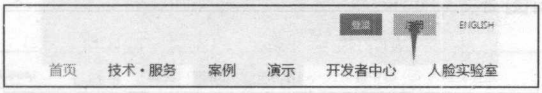



图 8-9　Face++注册页面

（3）注册完成后，登录自己的账号，单击“我的应用”选项卡，进入“我的应用”界面后，再单击“创建应用”按钮，如图 8-10 所示。

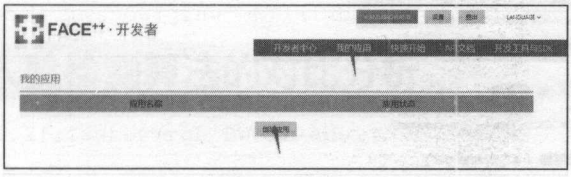



图 8-10　Face++创建应用界面

（4）创建应用完成后提交，会显示如图 8-11 所示的界面，从而可以获得 API_KEY 和 API_SECRET。之后可通过单击“我的应用”→“管理”，来查看 API_KEY 和 API_SECRET 的值，如图 8-12 所示。

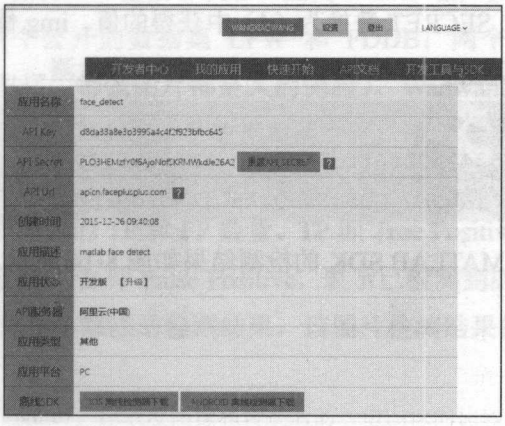



图 8-11　应用信息界面

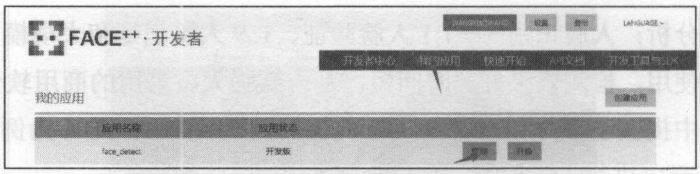



图 8-12　Face++应用列表界面

（5）下载 SDK。单击“开发工具与 SDK”选项卡，本书实验应用 MATLAB SDK 进行测试，如图 8-13 所示。

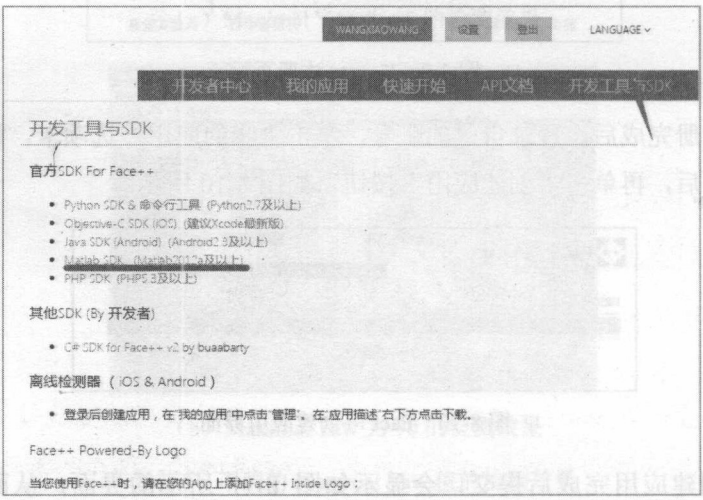



图 8-13　Face++SDK 下载界面

（6）下载好的程序为 facepp-matlab-sdk-master，打开 facepp_demo.m 文件，把 API_KEY 和 API_SECRET 替换为（4）中获得的值。img 修改为待检测的图片路径和名称，运行 facepp_demo.m 文件即可进行人脸检测。

```
img = 'demo.jpg';
API_KEY = 'd45344602f6ffd77baeab05b99fb7730';
API_SECRET = 'jKb9XJ_GQ5cKs0QOk6Cj1HordHFBWrgL';
```

使用 Face++的 MATLAB SDK 的检测结果如图 8-14 所示。



图 8-14　Face++的检测结果

上面展示了如何使用 Face++的 MATLAB SDK 进行人脸检测，Face++还有更多的 SDK 和强大的功能，读者可以查看 Face++的官方网站[2]。

## 8.3　各种人脸检测算法的对比分析

本书在前面章节已经介绍了 6 种人脸检测算法及两款商业软件，分别是 DPM、LAEO、CNN Facial Point Detection、Viola&Jones、DDFD 和 Fast R-CNN，以及 VeriLook 和 Face++。为了解这些算法的检测效果，本书进行了大规模的测试实验。该大规模测试的算法有 4 个，分别是 DPM、LAEO、CNN Facial Point Detection 和 Viola&Jones，不包括 DDFD，因为 DDFD 噪声太多。大规模测试的数据集有 4 个，两个公开的数据集 LFW 和 FDDB；两个自己收集的数据集 Wanwan1（100 张图片，每张图片都是人的侧脸）、Wanwan2（100 张图片，每张图片都是 360°人脸）。关于这 4 个数据集的介绍，请参考本书第 1 章的内容。

每个算法分别在这 4 个数据集上进行测试，大规模测试结束后，统计每个算法在各个数据集上检测后的 TP 和 FP 数量。TP 即 True Positive，表示已检测到的并且是人脸的图片数量；FP 即 False Positive，表示已检测到的但不是人脸的图片数量。图 8-15 所示为某个算法的检测结果，该图片检测结果的 TP=1，FP=1。

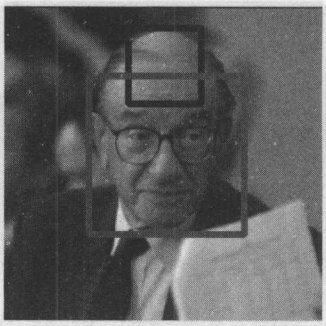


图 8-15　某个算法的检测结果

表 8-1 中统计了每个数据集上人脸的数量，以及每个算法在该数据集上检测到的 TP 数量和 FP 数量。

表 8-1　人脸检测结果对比

| 数据集 | | DPM | | LAEO | | Viola&Jones | | CNN Facial Point Detection | |
|---|---|---|---|---|---|---|---|---|---|
| 名称 | 人脸数量 | TP | FP | TP | FP | TP | FP | TP | FP |
| LFW | 601 | 534 | 61 | 450 | 26 | 513 | 12 | 526 | 0 |
| FDDB | 833 | 698 | 47 | 580 | 56 | 650 | 22 | 690 | 0 |
| Wanwan1 | 177 | 114 | 19 | 91 | 25 | 43 | 11 | 105 | 1 |
| Wanwan2 | 236 | 110 | 20 | 50 | 45 | 43 | 7 | 34 | 2 |

根据表 8-1，我们可以计算出每个算法在各个数据集上的检测率和检错率。表 8-2 统计了各个算法在 4 个数据集上检测完成所需的时间，以及在各个数据集上的检测率。表 8-3 统计了每个算法在各个数据集上的检错率。

例如，DPM 在 LFW 数据集上的检测率为：534/601×100%=88.85%，DPM 在 LFW 数据集上的检错率为：61/601×100%=10.15%。

表 8-2　各算法的检测率及运行时间

| 算法名称 | 运行时间 | LFW 的检测率 | FDDB 的检测率 | Wanwan1 的检测率 | Wanwan2 的检测率 |
|---|---|---|---|---|---|
| DPM | 140h | 88.85% | 83.79% | 64.44% | 46.61% |
| LAEO | 135h | 74.88% | 69.63% | 51.41% | 21.19% |
| Viola&Jones | 2.3h | 85.36% | 78.03% | 24.29% | 18.22% |
| CNN Facial Point Detection | 2.2h | 87.52% | 82.83% | 59.32% | 14.41% |

表 8-3　各算法的检错率汇总

| 算法名称 | LFW 的检错率 | FDDB 的检错率 | Wanwan1 的检错率 | Wanwan2 的检错率 |
|---|---|---|---|---|
| DPM | 10.15% | 5.64% | 10.73% | 8.47% |
| LAEO | 4.33% | 6.72% | 14.12% | 19.07% |
| Viola&Jones | 2.00% | 2.64% | 6.21% | 2.97% |
| CNN Facial Point Detection | 0% | 0% | 0.56% | 0.85% |

注：由于 LFW 数据集中有 13 233 张图片，FDDB 数据集中有 28 736 张图片，图片数量太多，在统计检测数量和检错数量时，本书只统计了 LFW 和 FDDB 数据集中的前 500 张图片。运行时间仍然是 4 个数据集全部运行完成所花费的时间。

综合表 8-2 和表 8-3 可知，DPM 在各个数据集上的检测率都是最高的，但是它的检错率也比较高，并且运行时间太长。所以，DPM 的优势为：适合各种角度的人脸检测，如正脸、侧脸、360°人脸；缺点为：运行时间长，是 CNN Facial Point Detection 的 64 倍。

CNN Facial Point Detection 在数据集 LFW、FDDB 和 Wanwan1 上的检测率仅比 DPM 低一点，但它的检错率几乎为 0，并且运行时间最短，2 个小时就能检测完将近 5 万张图片。所以，CNN Facial Point Detection 的优势为：检测时间短，对于正脸和侧脸检测效果好，错检率几乎为 0；缺点是：不适合检测 360°人脸。

Viola&Jones 在 LFW 和 FDDB 数据集上检测率一般，但在 Wanwan1 和 Wanwan2 数据集上检测率非常低。Viola&Jones 检测时间非常短，和 CNN Facial Point Detection 不相上下。所以，Viola&Jones 的优点为：适合检测正脸，检测时间短，检错率也较低；缺点是：不适合检测侧脸和 360°人脸。

LAEO 在各个数据集上的检测率都比 DPM 低，并且运行时间长，是 CCN Facial Point Detection 运行时间的 61 倍，和 DPM 的运行时间不相上下。所以，LAEO 的优点是：可以检测各个角度的人脸；缺点是：运行时间长，和其他算法相比，检测率低。

## 8.4　视频中的人脸检测与追踪

前面介绍的各种人脸检测算法都是基于静态图片的，本节我们将利用

Viola&Jones 检测器对视频中的人脸进行检测。对应的项目名称为 vedioDetect。运行该项目之前，只需在 Visual Studio 中配置好 OpenCV 视觉库即可，类似 Viola&Jones 检测算法的配置。

该程序首先读取视频中的每一帧，然后利用本书第 3 章中 Code7 的方法检测出每一帧中人脸的对应位置。下面，我们从源代码上面分析一下该程序是如何获得视频中的每一帧并进行检测的。

读者如果要修改检测视频的路径，则可在 vedioFace.cpp 中修改，如下所示。

```
VideoCapture capture("D://google//download//IMG_1560.MOV");
```

修改好路径后，运行 vedioFace.cpp 即可对该视频进行人脸检测。Code1 主要来自文献[3]，用于读取视频中的帧。

**Code1：vedioFace.cpp**

```
int main()
{
    CascadeClassifier cascade;//创建级联分类器对象
    string cascadeName = "haarcascade_frontalface_alt.xml";//训练好的人脸分类器
    if( !cascade.load( cascadeName ) )//加载级联分类器
    {
        cerr << "ERROR: Could not load classifier cascade" << endl;
        return 0;
    }
    double scale = 1.3;

    //打开视频文件：其实就是建立一个 VideoCapture 结构
    VideoCapture capture("D://google//download//IMG_1560.MOV");
    //检测是否正常打开:成功打开时，isOpened 返回 ture
    if(!capture.isOpened())
    {
        cout<<"fail to open!"<<endl;
        return 0;
    }
    //获取视频中的总帧数
    long totalFrameNumber = capture.get(CV_CAP_PROP_FRAME_COUNT);
```

```
        //设置开始帧
        long frameToStart = 0;
        capture.set( CV_CAP_PROP_POS_FRAMES,frameToStart);
        //设置结束帧
        long frameToStop = totalFrameNumber;
        if(frameToStop < frameToStart){
            cout<<"结束帧小于开始帧，程序错误，即将退出！"<<endl;
            return -1;
        }
        else{
            cout<<"结束帧为：第"<<frameToStop<<"帧"<<endl;
        }
        //获取帧率
        double rate = capture.get(CV_CAP_PROP_FPS);
        //定义一个用来控制读取视频循环结束的变量
        bool stop = false;
        //承载每一帧的图像
        Mat frame;
        int delay = 100/rate;
        //利用 while 循环读取帧
        long currentFrame = frameToStart;

        //滤波器的核
        int kernel_size = 3;
        Mat kernel = Mat::ones(kernel_size,kernel_size,CV_32F)/(float)
(kernel_size*kernel_size);

        while(!stop)
        {
            //读取下一帧
            if(!capture.read(frame))
            {
                cout<<"读取视频失败"<<endl;
                return -1;
            }
            //这里加滤波程序
            filter2D(frame,frame,-1,kernel);
```

```
        if( !frame.empty() )//读取图片数据不能为空
                {
                if(frame.channels()!=1)
                detectAndDraw( frame, cascade, scale);
        }
        namedWindow("image",WINDOW_NORMAL); %可以调整输出的窗口
        imshow("image",frame);
        int c = waitKey(delay);

        //按下 Esc 键或者到达指定的结束帧后退出读取视频
        if((char) c == 27 || currentFrame > frameToStop)
        {
        /       stop = true;
        }
        //按下按键后会停留在当前帧，等待下一次按键
        if( c >= 0){
                waitKey(0);
        }
        currentFrame=currentFrame+10;
    }
    //关闭视频文件
    capture.release();
    waitKey(0);
    return 0;
}
```

对视频中的每一帧，利用本书第 3 章 Code7 中的 detectAndDraw 函数进行人脸检测。读者运行上述程序后，可看到检测效果，这里不再展示。

# 参考文献

[1] VeriLook software. http://www.neurotechnology.com.

[2] Face++. http://www.faceplusplus.com.cn/.

[3] OpenCV 学习笔记. http://blog.csdn.net/thefutureisour/article/details/7530344.

人脸的智能识别在身份验证、交通、安防、金融等方面具有重要应用。本书旨在揭秘“刷脸”背后的技术，不仅包括人们经常听说的人脸识别技术，更有人脸检测、人脸关键点定位、人脸检索等相关技术。

**本书主要包括以下内容：**

1. 人脸识别概述及相关数据集的介绍，以及数字图像处理的基础知识。
2. DPM、LAEO、Viola&Jones、CNN Facial Point Detection、Deep Dense Face Detector(DDFD)、Fast R-CNN等人脸检测算法。
3. DeepID、VGG Face CNN Descriptor等当下流行的基于深度学习的人脸识别算法，以及传统的Eigenfaces、Fisherfaces等人脸识别方法。
4. PHash、DHash、Gabor、BoF 、HoG等常用的人脸检索方法，以及KD-Tree等加快人脸检索速度的方法。
5. VeriLook、Face++、OpenCV等知名人脸检测软件的介绍和应用示例。

对于人脸识别领域的初学者、工程师和研究生而言，本书是一本全面涵盖人脸检测、人脸识别、人脸检索等“刷脸”应用相关技术的重要参考书籍。

ISBN 978-7-121-32138-2

9 787121 321382 >

定 价：48.00元

策划编辑：董亚峰（dyf@phei.com.cn）
责任编辑：董亚峰
封面设计：田晨晨